micro:bot Kit Experiment Guide

Introducing the micro:bot Kit

The micro:bit is a great platform for learning how to build and program robots! Combining the micro:bit with the SparkFun moto:bit - micro:bit Carrier Board (Qwiic) creates a flexible, low-cost robotics platform for anyone from students getting started with the micro:bit to the engineer looking to quickly prototype or build a proof of concept.

The micro:bot kit v2.0 is the extension of that idea: build simple robots quickly that leverage the capabilities of the micro:bit while implementing peripheral sensors and motor functions with simple programming in the Microsoft MakeCode environment as a gateway into robotics.

Product Showcase: SparkFun micro:bot kit v2.0

What's Included in the Kit?
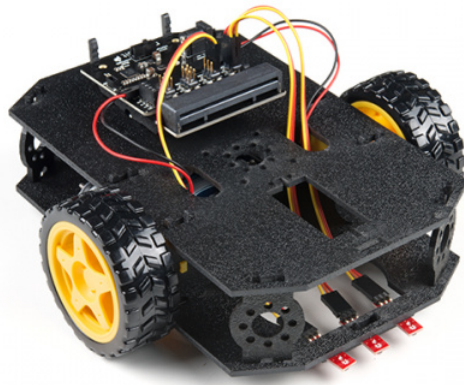
SparkFun micro:bot kit for micro:bit - v2.0
⊖ KIT-16275

The kit includes the following parts:

- **1x SparkFun moto:bit (Qwiic)** - Carrier board multiple I/O pins capable of hooking up servos, sensors and other circuits.
- **1x Shadow Chassis** - Our go to robotics chassis for tabletop robotics.
- **3x SparkFun Line Following Sensor** - Three sensors for detecting lines and nearby objects.
- **2x Hobby Servo Motors** - Program position with these motors.
- **2x Wheel — 65mm (Rubber Tire, Pair)** - Wheels to attach to the Hobby Motors.
- **3x Jumper Wire — 3-pin, 6"** - Wires to connect the line sensors to the moto:bit.
- **2x Hobby Gearmotor** - Motors for driving the robots wheels.
- **1x 4xAA Battery Holder** - Battery pack for powering the micro:bit and the motors.

Note: The kit does not include the following and they will need to be purchased these separately:

- **1x** micro:bit
- **1x** Micro-B USB Cable
- **4x** AA Batteries

USB micro-B Cable - 6 Foot
◉ CAB-10215

micro:bit Board
◉ DEV-14208

Panasonic Alkaline Battery - AA
⊙ PRT-15201

How to Use This Guide?

This guide is designed to get you started with the moto:bit board and the SparkFun micro:bot kit in a straight forward and simple way. We demonstrate each component's functionality and the corresponding code to make it work.

While you explore this guide, we urge you to take your time and tinker with the sensors, code, and the ideas shared to build something tailored to your application and creativity. Our goal is to get you enough information and know-how to make you dangerous and then release you into the wild to do whatever *you* do with *your* robot.

Be sure to share your projects with us over Twitter or Facebook! We are excited to see you Start Something!
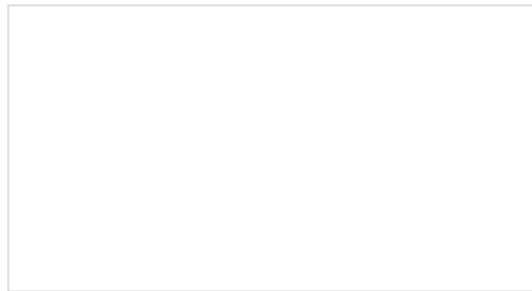
Suggested Reading

Before continuing with this guide, we recommend you be somewhat familiar with the concepts in the following tutorials:
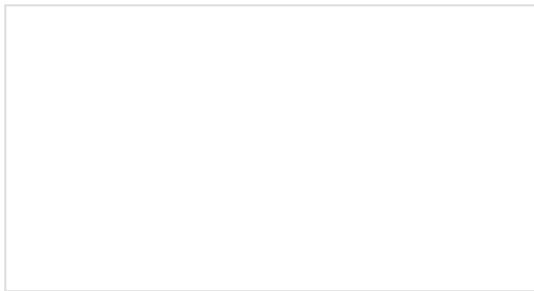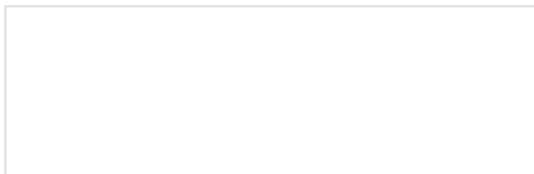

Accelerometer Basics
A quick introduction to accelerometers, how they work, and why they're used.

Hobby Servo Tutorial
Servos are motors that allow you to accurately control the rotation of the output shaft, opening up all kinds of possibilities for robotics and other projects.

Getting Started with the micro:bit

The BBC micro:bit is a compact, powerful programming tool that requires no software installation. Read on to learn how to use it YOUR way!

Open Source

All of our experiments and guides are licensed under the Creative Commons Attribution Share-Alike 4.0 Unported License. Feel free to remix and reuse our work. But please, share the love and give us attribution for our hard work!

To view a copy of this license visit this link, or write: Creative Commons, 171 Second Street, Suite 300, San Francisco, CA 94105, USA.

About the moto:bit Board

The moto:bit is a carrier board for the micro:bit. Similar to an Arudino shield, it is designed to add functionality to the micro:bit without the hassle of a number of other boards, soldering, and all of those jumper wires.
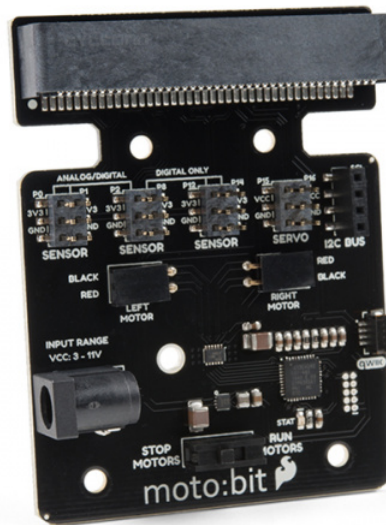
### SparkFun moto:bit - micro:bit Carrier Board (Qwiic)
DEV-15713

In this case, the moto:bit takes a micro:bit and turns it into a full blown robotics platform. Wondering what you can do with it? Well here are a few things...

- Control motors through an onboard H-Bridge
- Read digital sensors such as line and bump sensors
- Read analog sensors like light sensors, temperature sensors, etc

- Control servo motors
- I^{2}C port for extending functionality

That is a lot of options in terms of bells and whistles! Let's take a closer look at the board and go over each section.

> **ⓘ Note:** If you have the previous version of this board (DEV-14213), note that the functionality is, for all intents and purposes, the same. Version 2.0 just has an upgraded micro:bit connector and a Qwiic connector to bring this board into our Qwiic eco-system. You should be able to use the original moto:bit board in the same manner as the revised board is used in the rest of this experiment guide!
>
> | | |
> | :---: | :---: |
> | *Qwiic moto:bit [DEV-15713]* | *moto:bit [DEV-14213]* |
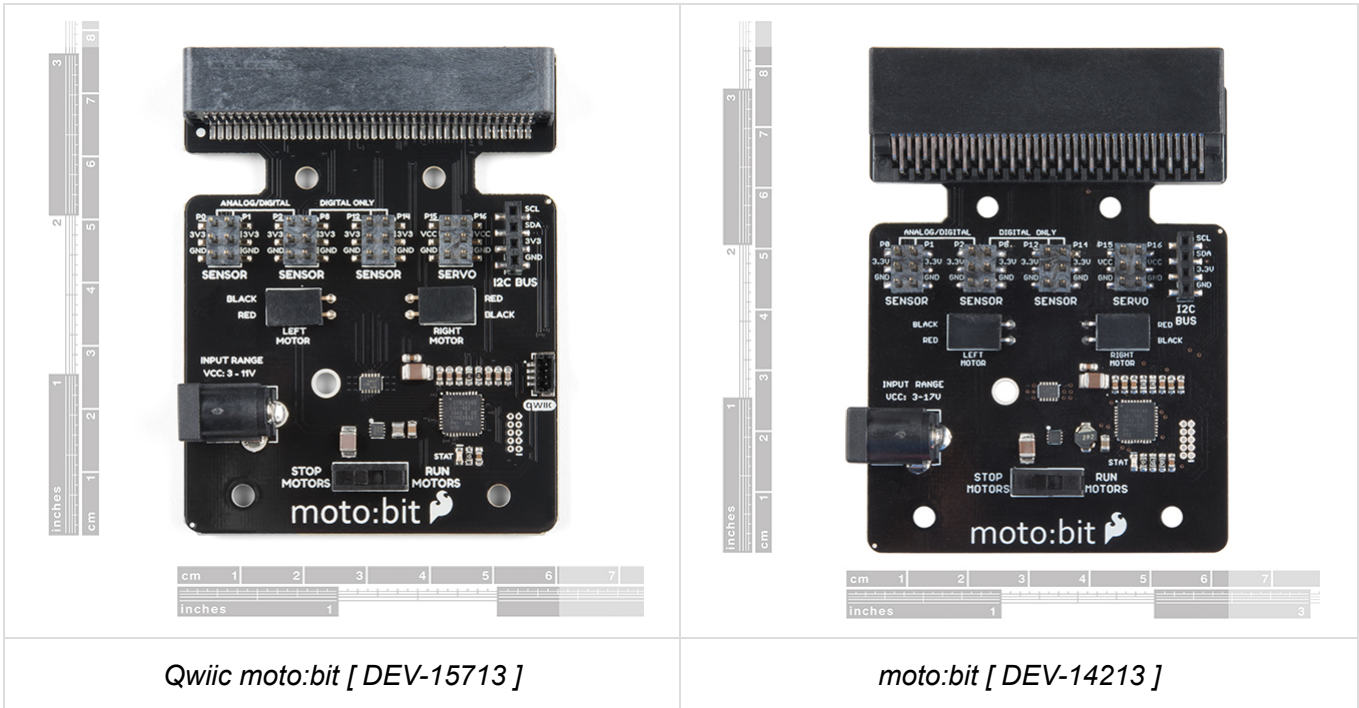
Edge Connector

The moto:bit connects to the micro:bit via an edge pin connector. This makes it handy to swap out micro:bits for programming, and it creates reliable connections to all of the different pins on the micro:bit.
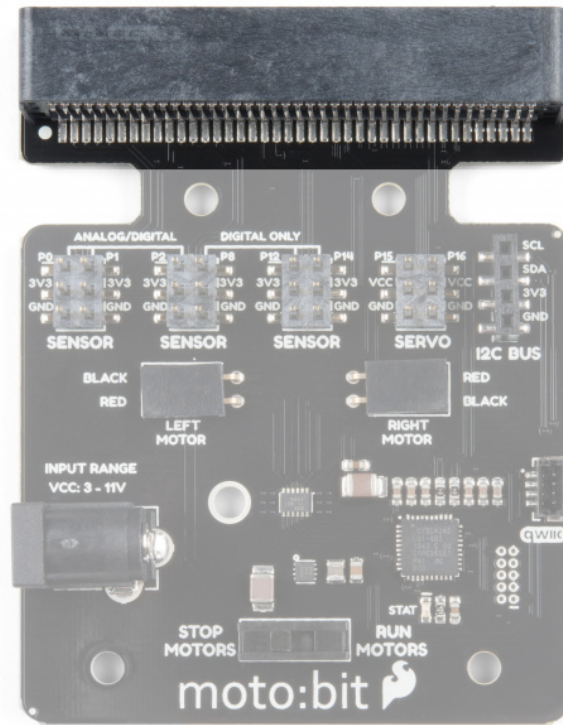
H-Bridge and Motor Pins

An H-Bridge is a chip that is the heart of a robot when it comes to driving motors and more specifically driving motors in both directions. Depending on the electrical state of specific pins on the H-Bridge, a motor drives forwards, backwards, and at different speeds. The good thing about this board is that if you are using Microsoft MakeCode, you actually don't really need to know a whole lot about the H-Bridge itself.
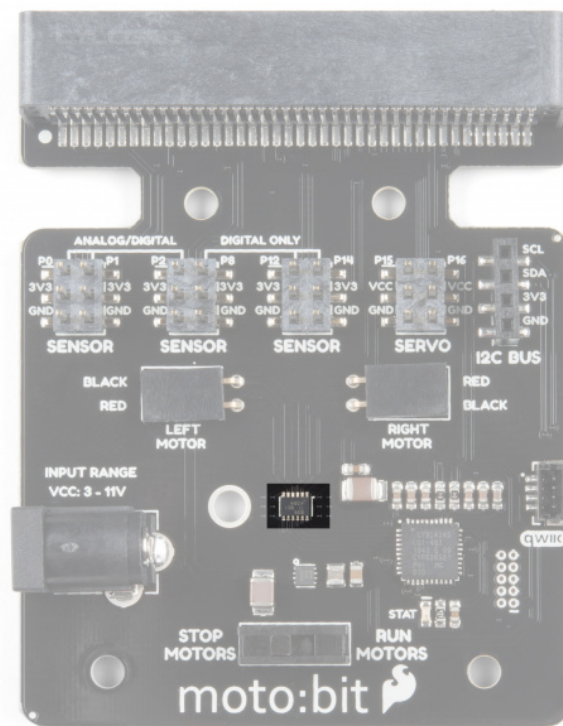

To connect the hobby motors that are included in the kit, you can insert them into the 2-pin female connectors just above the motor pins. The connectors are highlighted in the image below. Keep in mind that direction the hobby motors will move depends on the code to control the H-bridge motor driver, how the motors are attached to a

chassis, and the way the motors are wired to the input pins.
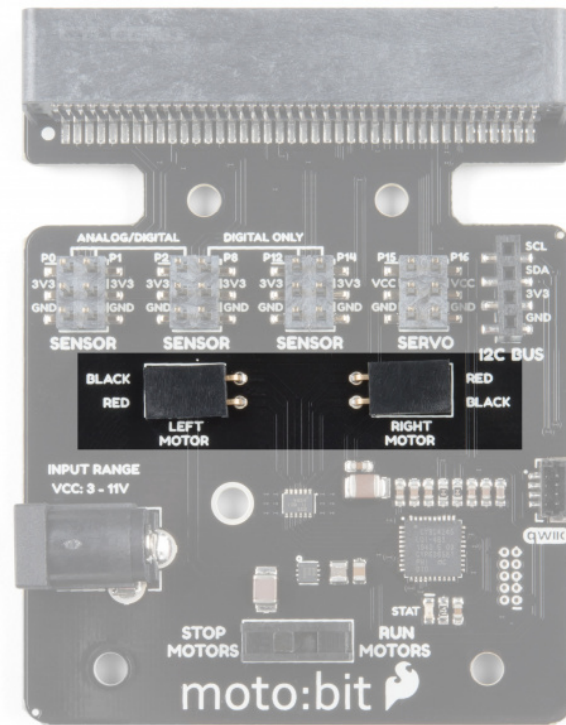
Motor Control Switch

The moto:bit has a switch that controls the power supply to the motors. That way you can have the robot powered while working on it or programming it and know that the robot is not going to start moving and drive off of the table. Believe us... that happens all the time!
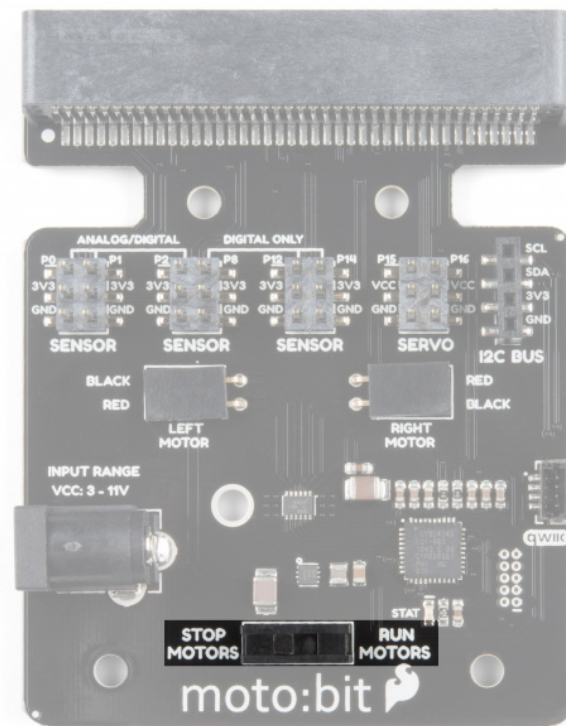
Input and Output Pins

The male pins on the moto:bit are for hooking up various inputs and outputs on your robot without using a breadboard to build elaborate circuits.
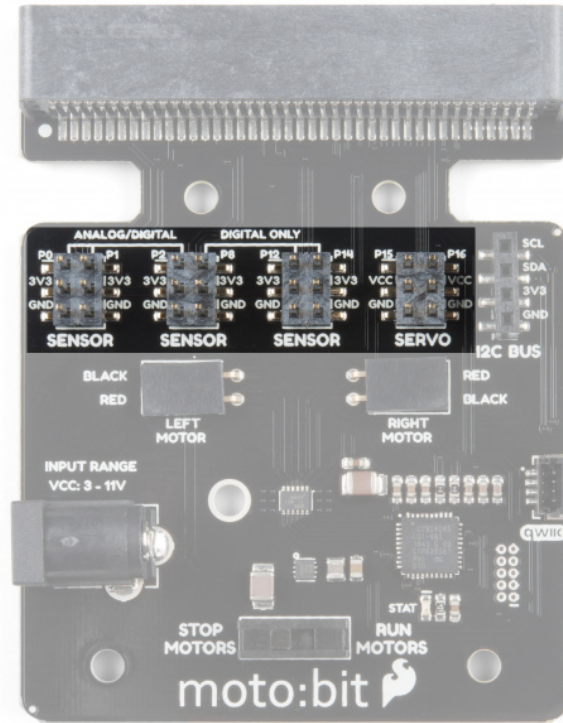
We have a number of sensors and actuators that are built in this pin formation and will work with this board.

SparkFun Line Follower Array
⊖ SEN-13582

Wheel Encoder Kit
⊖ ROB-12629

SparkFun RedBot Sensor - Mechanical Bumper
⊖ SEN-11999


SparkFun RedBot Buzzer
◉ ROB-12567

SparkFun RedBot Sensor - Accelerometer
⊙ SEN-12589

Servo Ports

No robot is complete without an arm, a swiveling "head," or some other type of movement other than wheels. Notice that a couple of the pin groups are designated as "Servo". You can connect servo motors directly to these pins and use them right out of the box with Microsoft MakeCode.
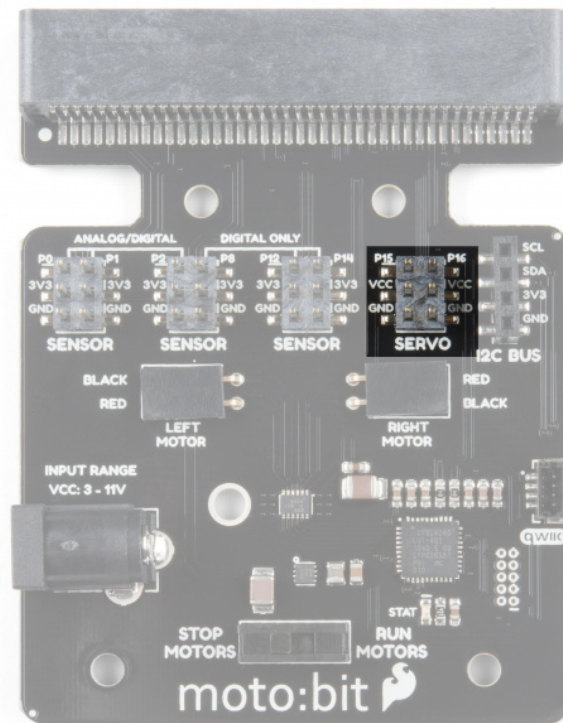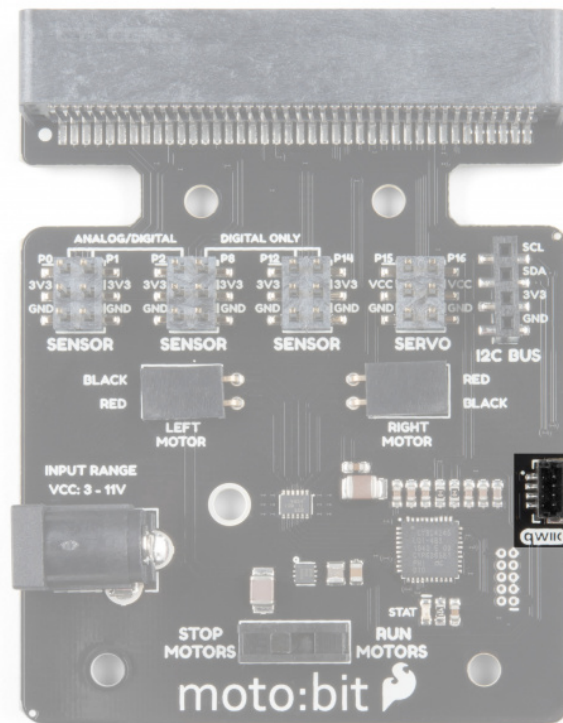

I²C Port

We broke out the I²C port of the micro:bit to an external port so that you can add any I²C capable sensor or actuator you can think of. It is standard pin arrangement to many of our I²C sensor breakout boards.
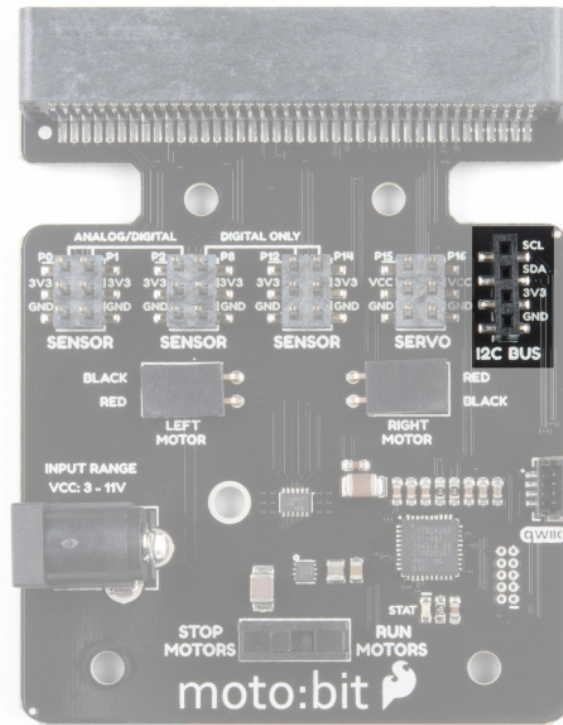
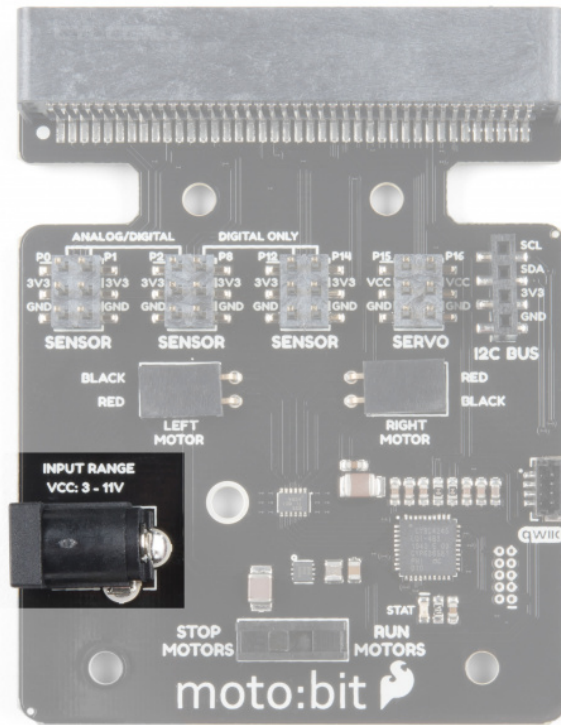## Qwiic Connector

With the updated moto:bit board, we've added a Qwiic connector so that our Qwiic line of products can be incorporated more easily.


Power

A standard barrel jack connector is used for easily powering your robot. We find that a 4xAA battery pack works great, but it will accept between **3V-11V** at the barrel jack. That's a whole lot of robo-power!

Maximum Input Voltage Range for Vcc: If you have the previous version of this board, the silkscreen indicates that the maximum input voltage range is between 3V to 17V. However, the actual voltage range that the board can support through the barrel jack is between **3V to 11V.** Please do not apply more than 11V to the power jack on the moto:bit. The updated version of this board lists the maximum input voltage correctly.
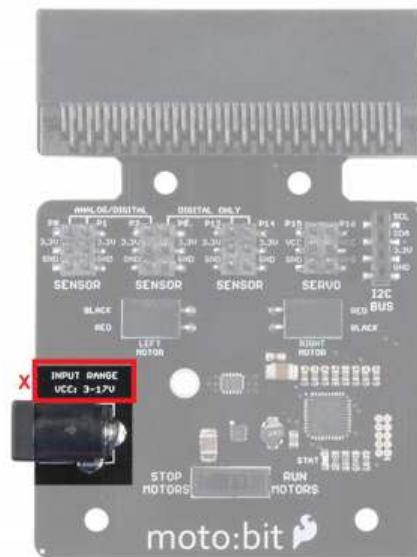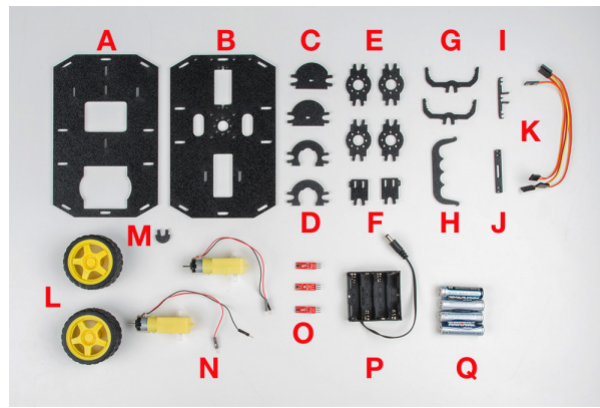

Assembling Your Robot

This section will cover how to assemble your robot chassis. ⏱**Assembly time: 30-60 minutes**

The robot chassis requires the following pieces to build. You'll find most of them in your SparkFun micro:bot kit.

> **Note:** Several of the parts need to be snapped out of the main chassis panels.

Click on the image for a closer look.

| Letter	Part	Qty
A	Bottom Chassis Plate	1
B	Top Chassis Plate	1
C	Front Motor Mount	2
D	Rear Motor Mount	2
E	Side Strut	4
F	Encoder Mount **(Not Used in this Guide)**	2
G	moto:bit Mount	2
H	Battery Pack Clip	1
I	Line Follower Mount	1
J	Line Follower Mount Plate	1
K	Jumper Wire — 3-pin, 6"	3
L	Wheels	2
M	Nub Caster	1
N	DC Motors	2
O	Line Follower Boards	3
P	4xAA Battery Holder	1
Q	4xAA Batteries (Not Included)	3

No Tools Necessary

The robot chassis does not require any additional tools.

> **WARNING:** Do not attempt to remove chassis parts by squeezing them with pliers. If you try to muscle it too much you risk breaking them and compromising the structure of your robot. Handle with care. ;)

A Note About Orientation

When we talk about the "front," "left," "right," and "back" of the Shadow Chassis, we are referring to specific sides of the robot when viewed from above.
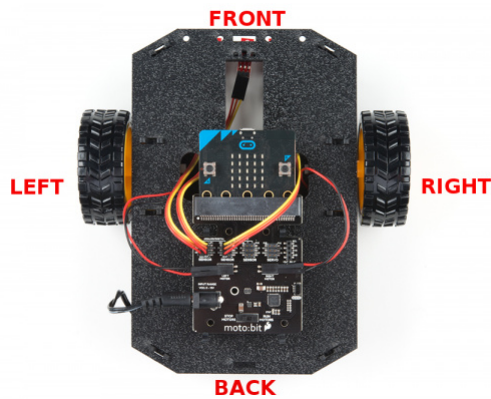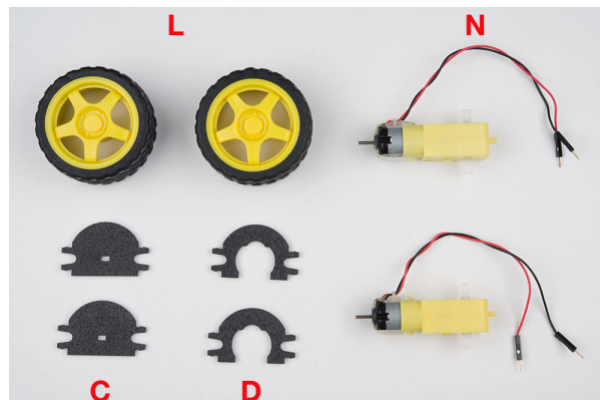

Notice that we consider the SparkFun moto:bit to be on the "back" of the bot and the Bumper Whiskers and Line Follower Boards to be in the "front."

Assembly

Installing Motors

Let's begin by installing the motors that will control the wheels. Locate the following:
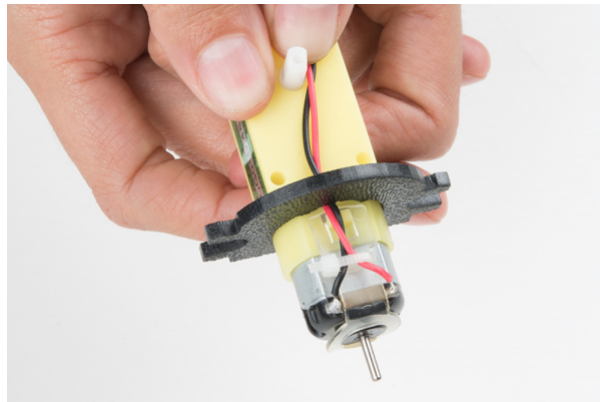
Attach Rear Motor Mounts

Hold the wires near the middle of the Motor (N), and carefully slide a Rear Motor Mount (D) in from the side and over the two motor wires. Be careful not to snag the wires, the cable tie, or the clear plastic strap.

Holding the motor wires, gently twist the Rear Motor Mount counter clockwise so that it snaps in place on the motor and the wires are centered in the gap of the motor mount. Again, be sure not to snag the wires under the motor mount.
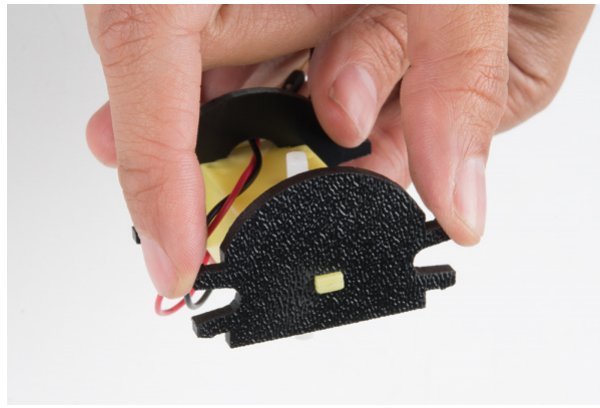

Repeat the process for the second motor, ensuring that it is a mirror image of the first.
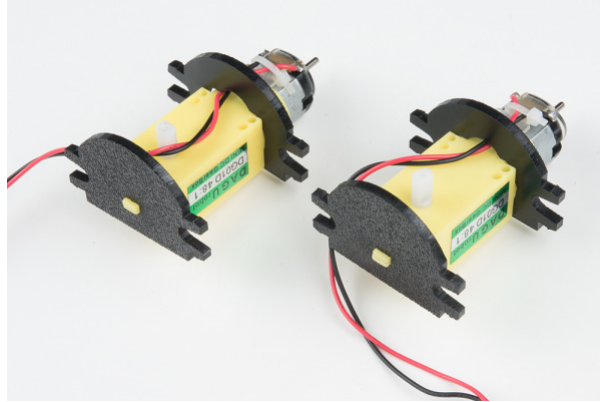

Attach the Front Motor Mounts

Slide a Front Motor Mount (C) onto the protruding eyelet on the front of a Motor (N). Ensure the rounded sides of the motor mounts are facing the same way.

Repeat the process for the second motor.

Attach the Motor Assemblies to the Chassis

Snap one of the motor assemblies into the left two horizontal slots of the Bottom Chassis Plate (A). Make sure that the rounded edges of the motor mounts and the wires are facing toward the center of the chassis. Repeat for the opposite motor.
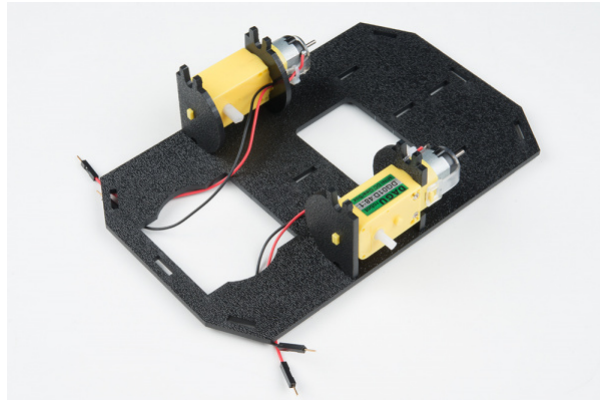
Attach the Wheels

Slide one Wheel (L) onto the plastic shaft of a Motor (N). Look at the motor shaft. Notice it has two flat edges. Make sure to line up the flat edges of the motor shaft with the flat edges of the wheel.
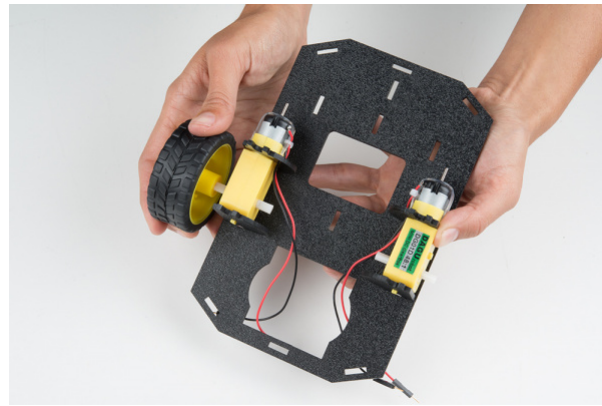
Repeat with the other wheel.

Installing the Line Sensors

This section will cover the Line Following Sensors array assembly. You'll first build the Line Following array and then attach it to the chassis. Locate the following:
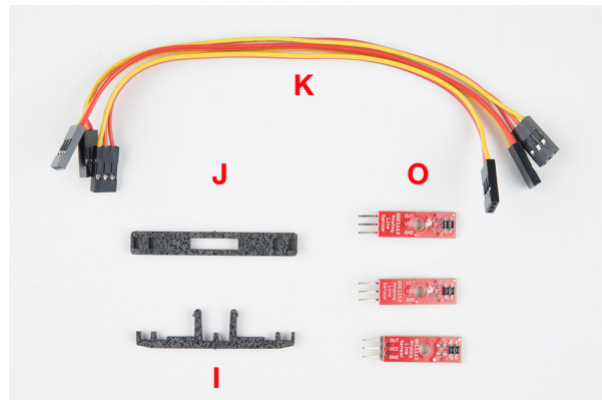
Construct the Line Follower Assembly

Attach the three Line Follower Boards (O) to the Line Follower Mount (I) such that the rectangular pegs in the Line Follower Mount poke through the mounting holes in the Line Follower Boards. Make sure the sensors are facing *away/down* from the clip of the mount.
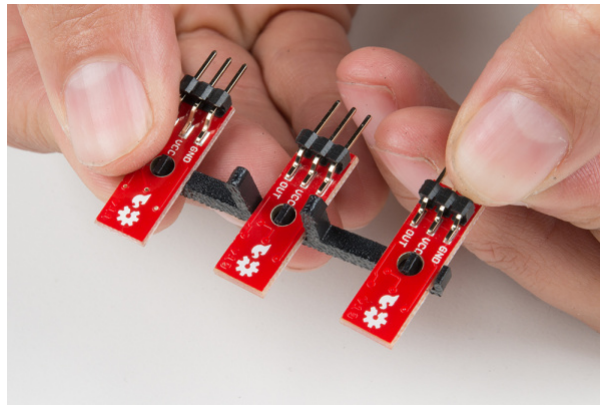
Place the Line Follower Mount Plate (J) on top of the Line Follower Mount (I) so that the center clip of the mount is poking through the center slot of the plate.
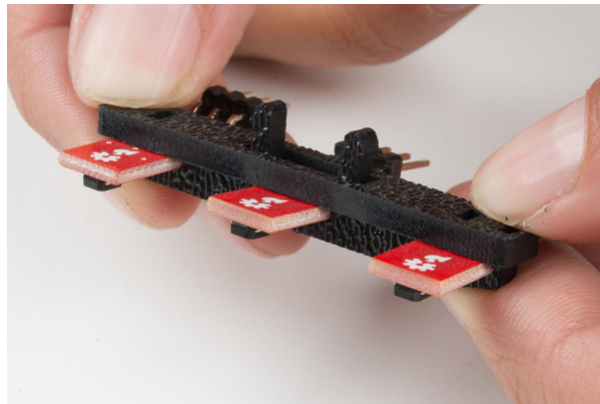
Attach the Cables

You will need to connect a 3-Wire Jumper Cable (K) to each of the Line Follower Boards (O). Note the color of the wire attached to each pin.

Line Follower Connections:

Jumper Wire Color	RedBot Sensor - Line Follower
Red	GND
Orange	VCC
Yellow	OUT

Attach all 3 cables to the 3 Line Follower Boards. Notice that the yellow wire should be on the right (out) and the red wire should be on the left (ground).
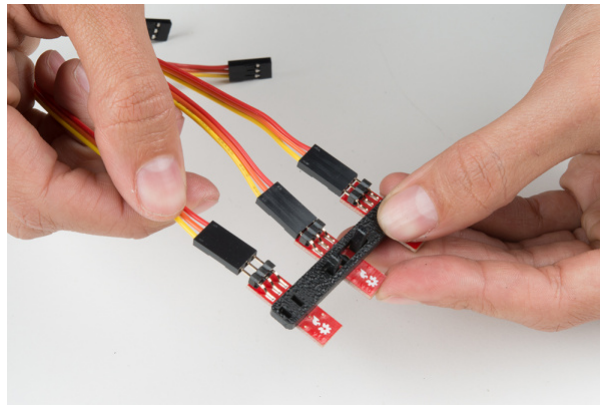
Attach the Line Follower Assembly to the Chassis

Locate the wide, rectangular slot near the front of the chassis and snap the line follower assembly in from the bottom side of the chassis. Route the cables through the large hole in the bottom plate.
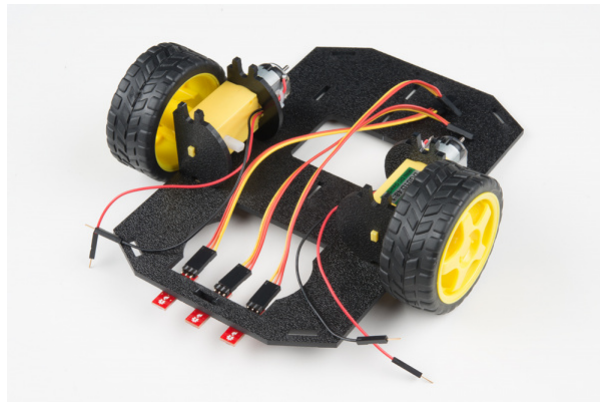
The bottom of your chassis should look like the following image allowing the line sensors to be facing down.
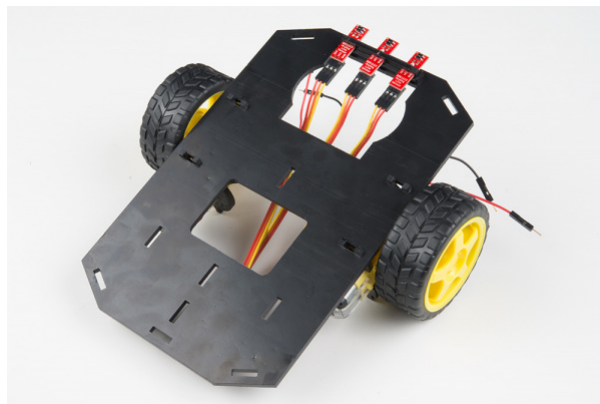
Final Assembly

With the motors and a few sensors attached, we can assemble the main body of the robot. Locate the following:
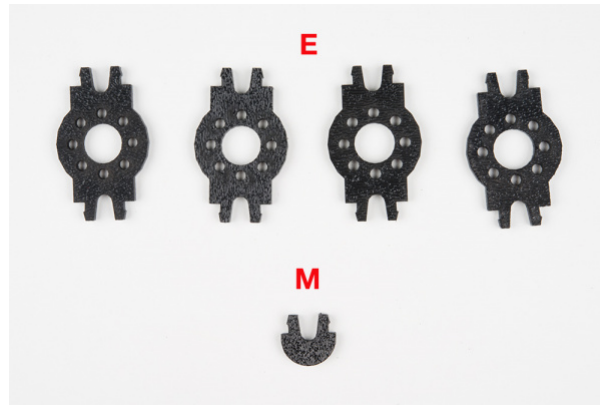
You will also need the **Top Chassis Plate** and **Bottom Chassis Plate** assemblies, which have any additional parts and sensors you attached in previous steps.

Attach the Nub Caster

Snap the Nub Caster (M) into the slot on the back of the Bottom Chassis Plate assembly. Make sure the Nub Caster is on the side opposite the motors (the bottom side).
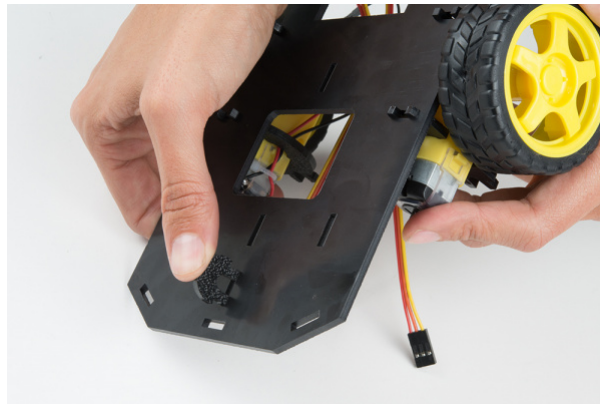
Add the Side Struts

Snap the four Side Struts (E) into the diagonal slots on the four corners of the Bottom Chassis Plate assembly.

Pull the cables through the cutouts on the chassis.

Route the Cables

Position the Top Chassis Plate over the Bottom Chassis Plate -- but do not snap the two plates together yet. Make sure that the front sides of each plate line up.

Route the wires and cables through the left and right oval slots in the Top Chassis Plate assembly as shown. For the center line follower sensor, route this cable through the left oval slot.

> **NOTE:** It might be a good idea to use some pieces of masking tape to mark which cables go to which component. It's not necessary, but it might help keep things organized.

Cable Routing:

Cable Connection	Oval Side
Left Line Follower	Left
Center Line Follower	Left
Right Line Follower	Right
Left Motor Wires (red and black)	Left
Right Motor Wires (red and black)	Right
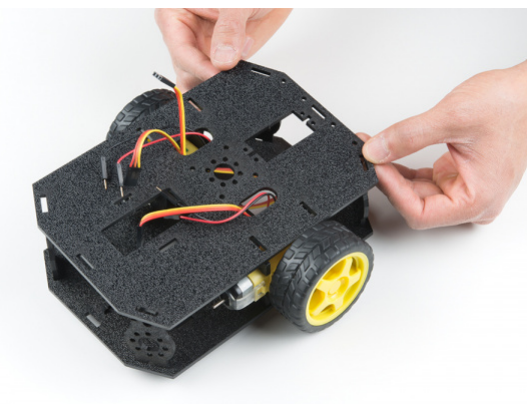
Attach Top Chassis Plate Assembly

Line up the Top Chassis Plate on top of all the struts, and carefully snap the Top Chassis Plate assembly onto the side struts and motor mounts. Press gently above each side strut individually until they each snap into place. If you have the Bumpers installed, make sure the boards are between the top and bottom plates.
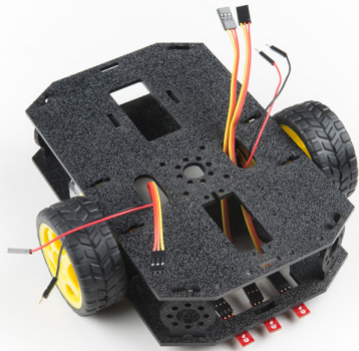
If you need to remove the plate to change anything, gently pull upward on each side strut individually. Do not attempt to use pliers or hand tools, or you may end up snapping the plastic clip.

Attaching the moto:bit

In this section, you will add brains of the robot: the micro:bit and SparkFun moto:bit. Locate the following:
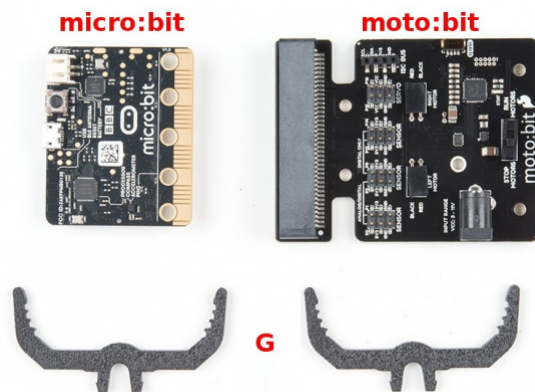
You will also need the full chassis assembly, which contains any additional parts you attached in previous steps.

Attach the moto:bit Mounts

Snap the two moto:bit mounts (G) into the vertical slots in the back of the top chassis plate near the large rectangular opening.
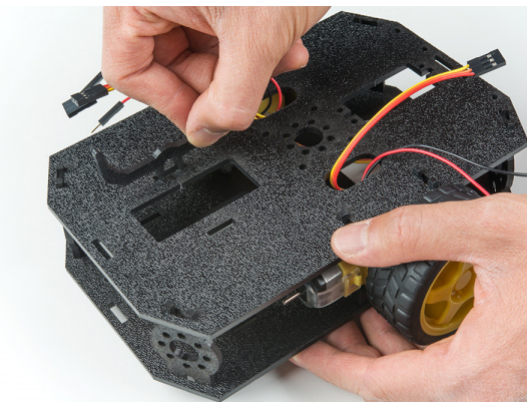
Both of these mounts will enable your moto:bit board to be mounted to the chassis.
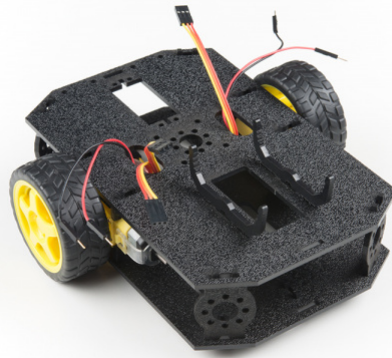
Add the moto:bit

Before you snap in the moto:bit board, insert your micro:bit into the moto:bit as shown here.
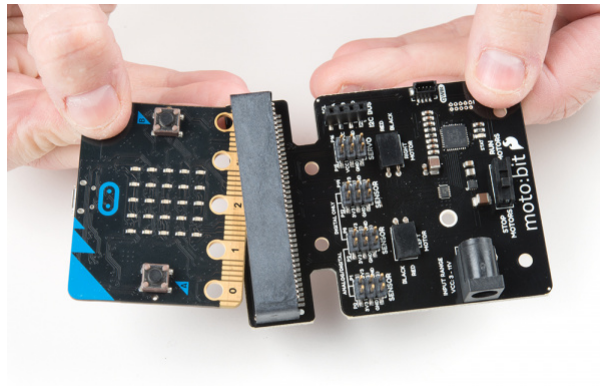
The moto:bit snaps into the lowest of the notches on the moto:bit Mounts (G). Make sure the power jack is facing the left side of the robot. Push gently and evenly until it snaps into place.
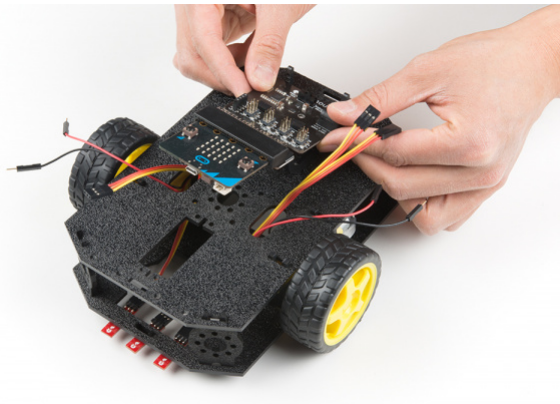
Note: The other slots in the moto:bit mounts can be used to hold the Arduino Uno or the Sparkfun RedBoard.

Once snapped, your setup should look like the image below without the wires connected to the moto:bit board
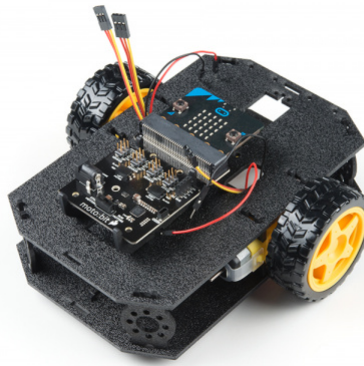
Connecting the Cables

It is time to connect the jumper wires; it is really important that these connections are right.
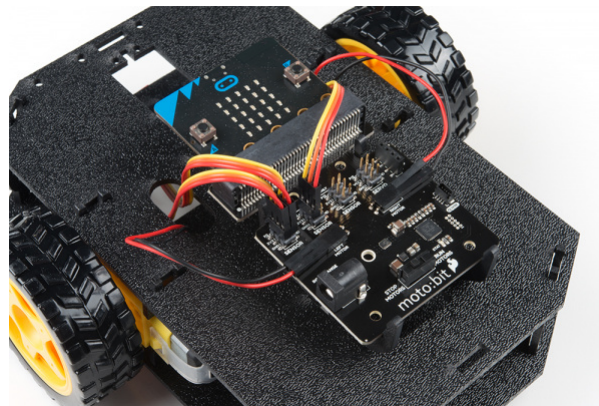
Follow along with the tables and annotated image with the cables connected for reference. Trace each cable poking through the top chassis plate to make sure you know what it is connected to.
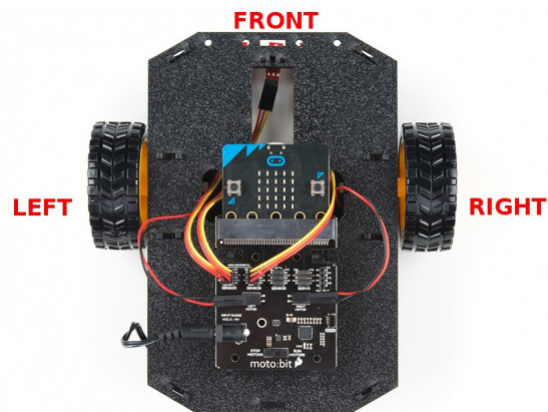
Please Note: When you have the micro:bot upright and the front of the chassis facing away from you, "left" sensors/motors will be on the left side and "right" sensors/motors will be on the right side. Also, the motor wires are intentionally switched for the right motor -- see notes below.

Line Followers

Below is a table for each connection between moto:bit and analog line follower. Start with the left line follower.
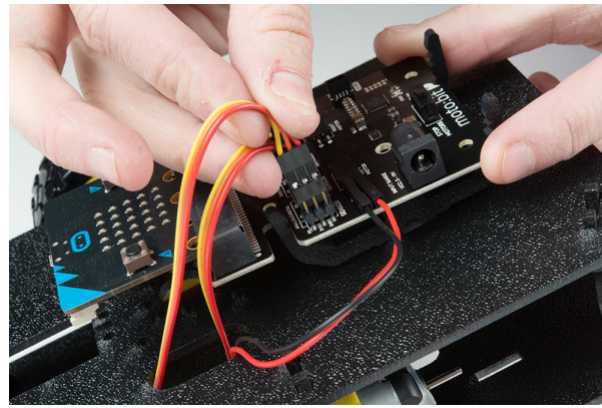
Left Line Follower:

SparkFun moto:bit Pins	Jumper Wires	Left Line Follower Board
P0	3-Wire Jumper Cable - Yellow	OUT
3.3V	3-Wire Jumper Cable - Orange	VCC
GND	3-Wire Jumper Cable - Red	GND

Center Line Follower:

SparkFun moto:bit Pins	Jumper Wires	Center Line Follower Board
P1	3-Wire Jumper Cable - Yellow	OUT
3.3V	3-Wire Jumper Cable - Orange	VCC
GND	3-Wire Jumper Cable - Red	GND

Right Line Follower:

SparkFun moto:bit Pins	Jumper Wires	Right Line Follower Board
P2	3-Wire Jumper Cable - Yellow	OUT
3.3V	3-Wire Jumper Cable - Orange	VCC
GND	3-Wire Jumper Cable - Red	GND

Once connected, the wires should look similar to the image below.
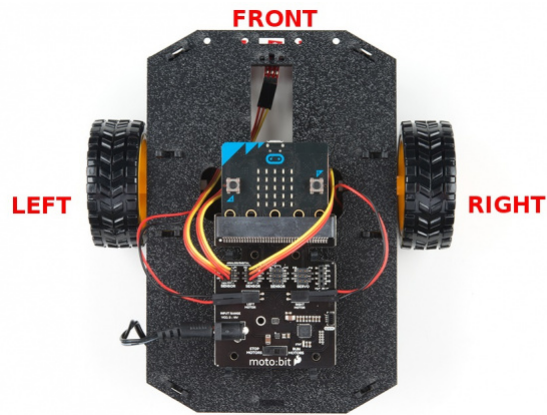
Motors

Choose a pair of motor wires to begin wiring up to the moto:bit. The way the motors are wired up to the moto:bit will affect how the micro:bot drives. Make sure to follow the color of the wire with the silkscreen on the board.
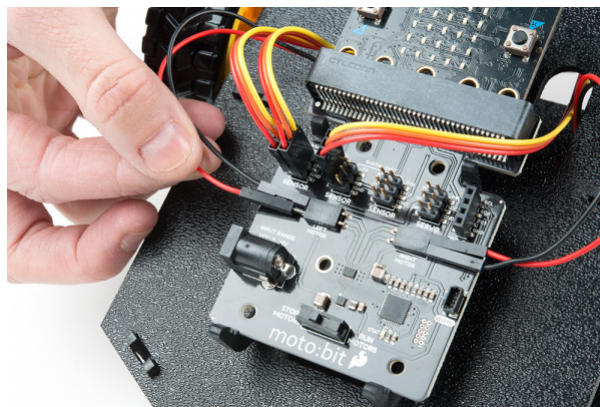
Below is a table for each connection between moto:bit and motor.

Left Motor:

SparkFun moto:bit Pins	Left Motor Jumper Wires
LEFT MOTOR - RED	Soldered on Motor Jumper Wire - RED
LEFT MOTOR - BLACK	Soldered on Motor Jumper Wire - BLACK

Right Motor:

moto:bit Pins	Right Motor Jumper Wires
RIGHT MOTOR - RED	Soldered on Motor Jumper Wire - BLACK
RIGHT MOTOR - BLACK	Soldered on Motor Jumper Wire - RED

Once all wires are connected it should look like the following image.

FRONT

LEFT   RIGHT
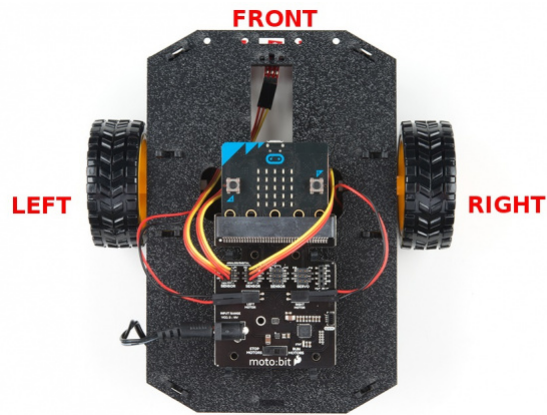
⚠ **Warning!** So what happens when you wire the motors the other way? Well, your motor may move a different direction when using the example code provided in this tutorial. There are two options if you notice the micro:bot moving in the wrong direction. You can flip the wiring of your motors or use the adjust the code blocks to change what is forward vs reverse. It's relative so either connection can be "correct" if the code you are using is consistent. We recommend following the silkscreen to reduce the amount of time troubleshooting.
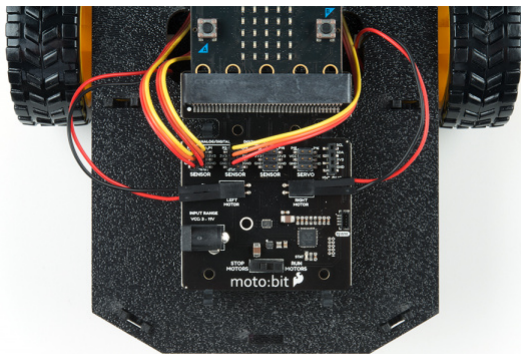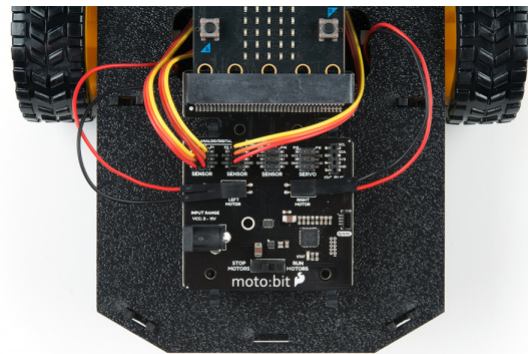
"Correct" Motor Wiring	*"Incorrect" Motor Wiring*

Batteries

The last step is to provide a power source for the micro:bot. You will need to provide your own AA batteries (Q). Locate the following:

H   P   Q
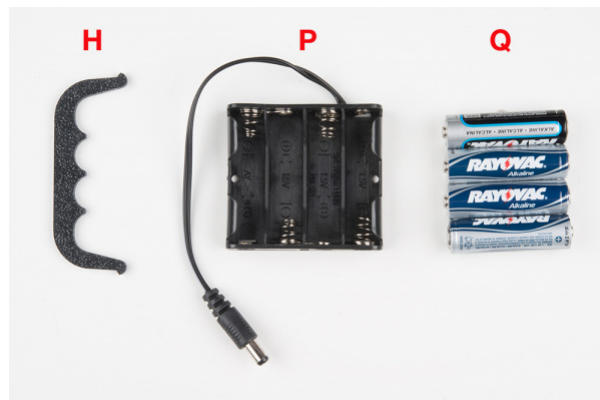
Insert Batteries

Insert the AA batteries into the Battery Holder (P). Makes sure the batteries are facing the correct direction, as per the markings inside of the Battery Holder.

Attach Battery Pack

Insert the Battery Holder (P) with batteries into the back cavity of the chassis. Position the Battery Holder so that the barrel jack cable comes out on the left side of the robot.
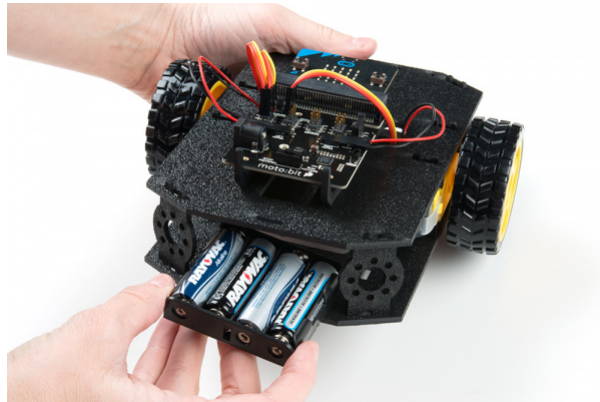
Insert the Battery Pack Clip (H) on top of the battery pack.
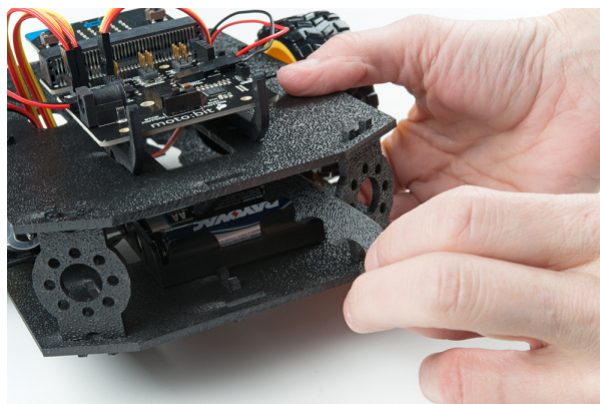
Twist and position the clip so that it rests on top of the battery pack.
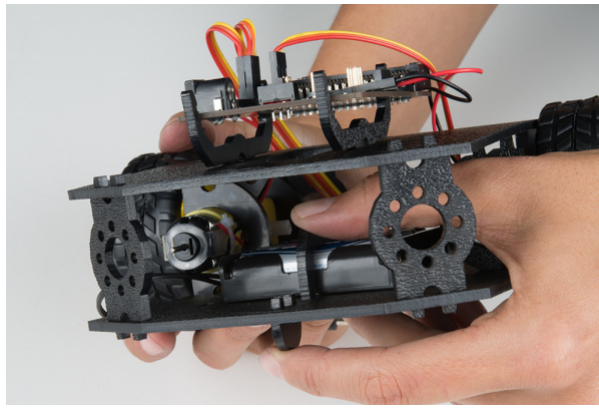
Push the clip down into the vertical slots in the Bottom Chassis Plate so it snaps in place.
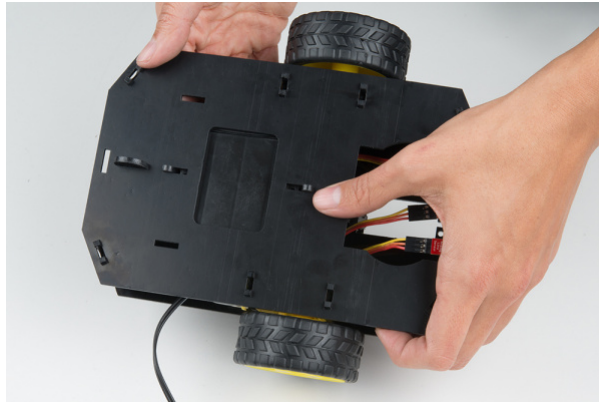
Route the barrel jack cable out of the left side of the chassis and up to the moto:bit.
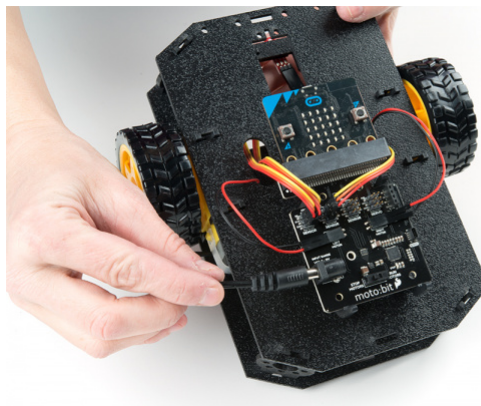
Plug the barrel jack cable into the barrel connector on the side of the moto:bit carrier board.
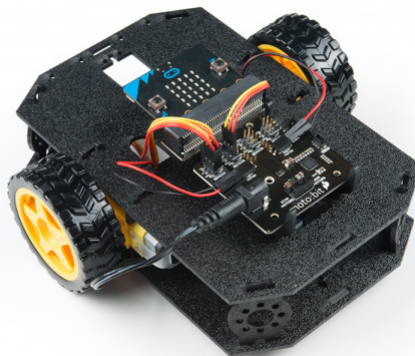
Changing the Batteries

If you find that you need to replace the batteries in the micro:bot, the process is simple. Unplug the battery pack from the moto:bit.
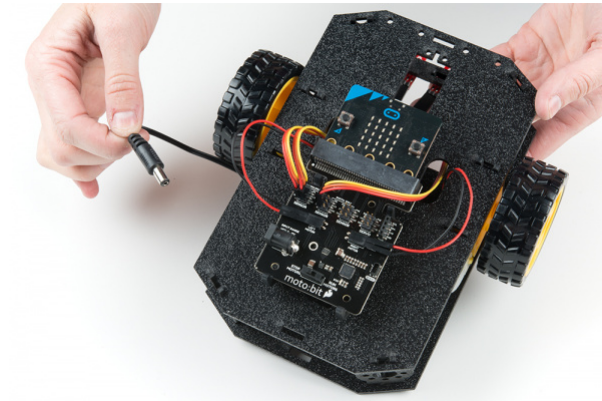
Turn the micro:bot over and push on the Battery Holder through the hole in the Bottom Chassis Plate. This will cause the Battery Pack Clip to unsnap from the Bottom Chassis Plate.

Slide the Battery Pack and Clip out from the back of the micro:bot.
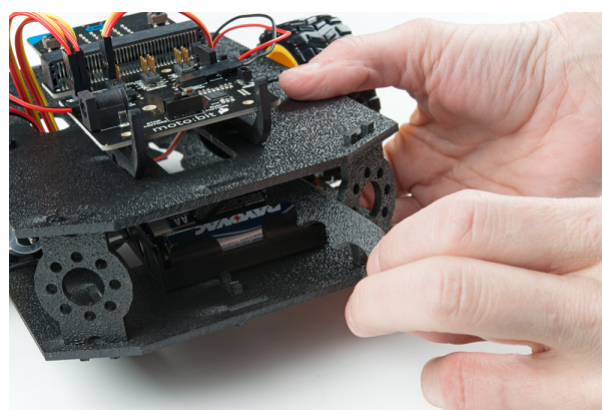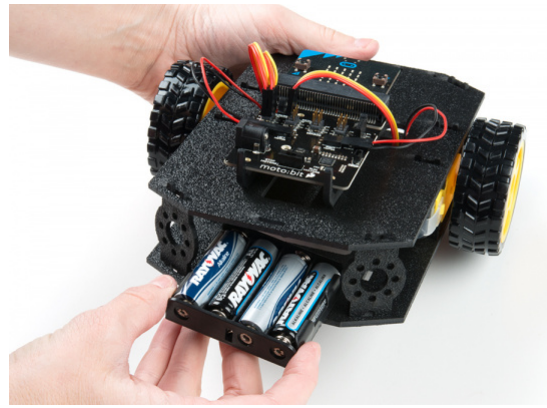
Change the batteries, and follow the steps in *Attach Battery Pack* section above to put the Battery Pack back in the micro:bot.

Installing the moto:bit Extension in MakeCode

To make the most out of the moto:bit with the least amount of coding, use the Microsoft MakeCode extension we wrote for the moto:bit board.

Extension

If you have used Arduino before, you probably know about a thing called a library; which is a collection of code that extends the functionality of the core programming language. MakeCode extension work the same way.

There are some amazing differences between Arduino libraries and MakeCode extensions. One of them is that MakeCode extensions include JavaScript functions, which you can use when programming in text, as well as all of the blocks you need to program using the block method. This makes learning and using new extensions straightforward and shortens the time to awesome when setting out to build the project of your dreams.

> **Heads up!** Previously, these libraries were referred to as MakeCode packages. They are now referred to as MakeCode extensions.

Installing a MakeCode Extensions

To install or add a new extension to your MakeCode toolbox (the list of different block groups), click on "Advanced" and then on "Add Extensions." This should be the last item on the list.
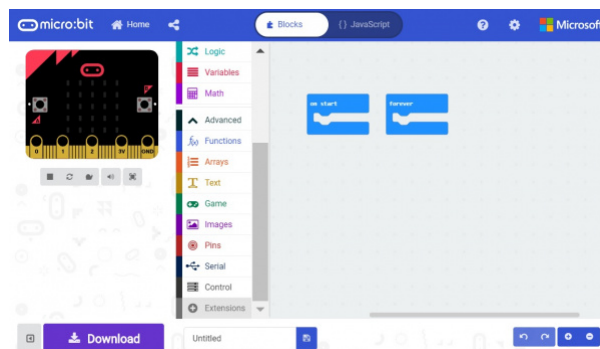
From here you can search for "**SparkFun**" or "**SparkFun moto-bit**," and it should show up as a public extension in the list. Go ahead and click on it.
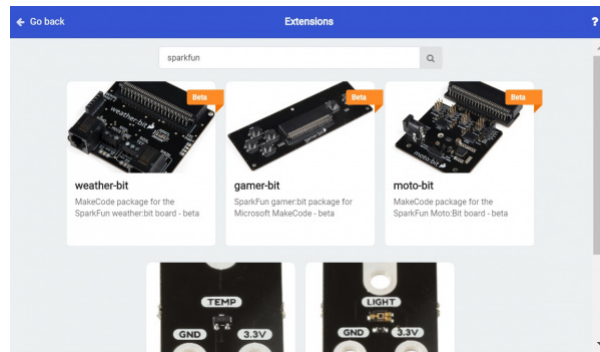
This will add all of the blocks to your toolbox. In general, this is a bit tricky as, depending on how the extension was written, it may either have its own toolbox or just add blocks to the existing ones. Take a look at your toolbox; for the moto:bit you should see...
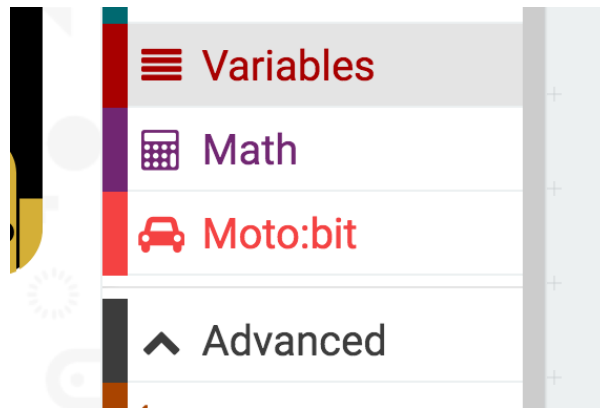
Great! You have now installed the moto:bit extension and are ready to use the board as well as the components that come in the micro:bot kit. As a side note, for every new MakeCode project that you make, you will have to load extensions over again. Not a big deal, but noteworthy! Now, let's put your extension to good use and get your robot moving!!!

Experiment 1: Driving and Turning

Introduction

The "Hello World" for the robotics world is getting your robot moving! Driving forwards, backwards, and steering is goal #1 when you are dealing with wheeled / moving robots. This experiment will cover just that: getting your robot moving using our moto:bit software package for Microsoft MakeCode. Once you get through this first experiment you will be golden!

Parts Needed

You will need the following parts:

- **1x** micro:bit Board *(Not Included with Kit)*
- **1x** Micro-b USB Cable *(Not Included with Kit)*
- **1x** moto:bit Carrier Board
- **2x** Wheels
- **1x** Assembled Shadow Chassis
- **2x** Hobby Gear Motors
- **1x** 4xAA Battery Holder
- **4x** AA Batteries *(Not Included with Kit)*

Didn't get the kit? Have no fear! Here are the parts you will need to complete this experiment. You may not need everything though depending on what you have. Add it to your cart, read through the guide, and adjust the cart as necessary.

Hobby Gearmotor - 140 RPM (Pair)
◉ ROB-13302

USB micro-B Cable - 6 Foot
◉ CAB-10215

micro:bit Board
◉ DEV-14208

SparkFun moto:bit
○ DEV-14213

Wheel - 65mm (Rubber Tire, Pair)
◉ ROB-13259

Battery Holder - 4xAA to Barrel Jack Connector
◉ PRT-09835

Shadow Chassis

⊙ ROB-13301

Panasonic Alkaline Battery - AA

⊙ PRT-15201

Suggested Reading

Before continuing on with this experiment, we recommend you be familiar with the concepts in the following tutorial:

Motors and Selecting the Right One
Learn all about different kinds of motors and how they operate.
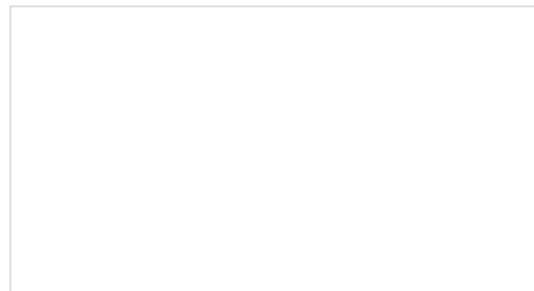
Getting Started with the micro:bit
The BBC micro:bit is a compact, powerful programming tool that requires no software installation. Read on to learn how to use it YOUR way!

Introduction to the Motors and Controlling Them

Motors take electrical energy and turn it into mechanical energy through putting a current through a coil of wire that creates a magnetic field. This then interacts with magnets in the motor, causing a push / pull effect. If you do that in a proper spacing and timing, then you can spin a shaft.

The hobby motors used with the micro:bot operate in a similar was as mentioned above, but they have a bit of help; a gear box. The ratio of the gearbox is 120:1, which means that for every 120 rotations of the motor shaft you get one rotation at the end of the gear box. This is a good thing because it gives our robot enough strength (torque) to turn wheels. In this case, our gearbox also changes the axis of the shaft so that we can easily use them to drive our robot.

Hardware Hookup

You should have already hooked up the motors during the assembly process of putting the robot together. But, we have a couple of things that we would like you to double check!
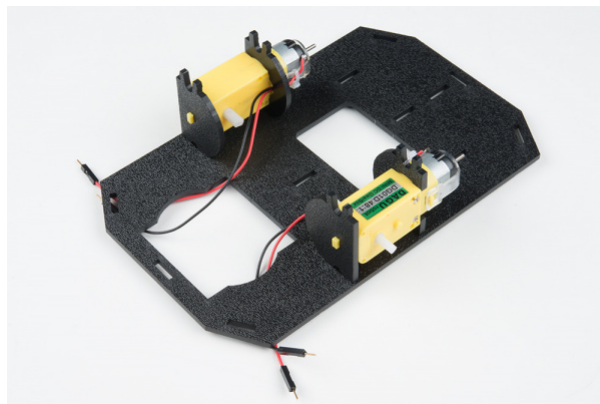
First of all, make sure that your red and black wires are plugged into your moto:bit and that they are in the correct ports. If you look closely the right and left ports are opposite from one another.
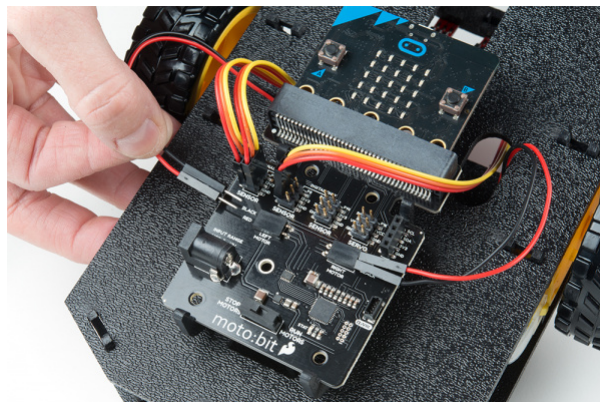
Why is that? Well, in robotics the right and left motors are mirrors of one another, which means they go in opposite directions from one another to go forward, hence the ports being backwards. If you plug them in as the same direction and tell both motors to spin in the sam direction, your robot would spin in a circle rather than drive forward! Weird, but true.

Running Your Script

We are going to use Microsoft MakeCode to program the micro:bit. Please open a browser window and navigate to makecode.microbit.org. This should open the MakeCode environment that you used to install the moto:bit package in.

> **Did you install the moto:bit package?** To add the moto:bit package as instructed in the Installing the moto:bit Package in MakeCode section of this tutorial.
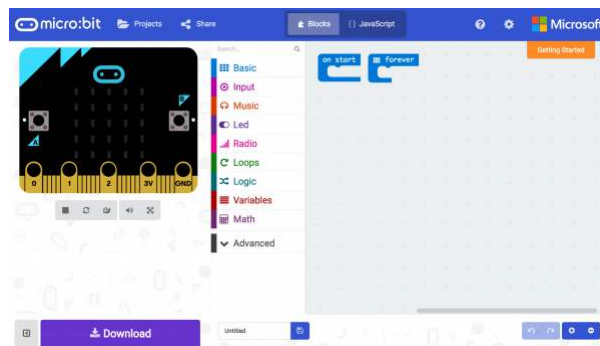
> **Note:** If this is your first time programming a micro:bit please be sure to see our Getting Started with the micro:bit tutorial on how to get your MakeCode program onto your micro:bit.

Now, you can either download the following example script below and drag and drop it onto your micro:bit, or use it as an example and build it from scratch in MakeCode.

▶ Simulator   🧩 Blocks   {} JavaScript   ☑ Edit

```
move  left ▽  motor motor  forward ▽  at  100
```

Code to Note

Let's take a look at the code and what to expect.

```
on start
    🚗 set left ▼ motor invert to  false ▼
    🚗 set right ▼ motor invert to  false ▼

⊙ on button A ▼ pressed
    🚗 turn motors ON ▼
    🚗 move left ▼ motor forward ▼ at  100  %
    🚗 move right ▼ motor forward ▼ at  100  %
    ▦ pause (ms)  1000
    🚗 move left ▼ motor reverse ▼ at  100  %
    🚗 move right ▼ motor forward ▼ at  100  %
    ▦ pause (ms)  200
    🚗 move left ▼ motor forward ▼ at  100  %
    🚗 move right ▼ motor forward ▼ at  100  %
    ▦ pause (ms)  1000
    🚗 turn motors OFF ▼
```

On Start

The code starts by setting each motor channel in the `on start` code block. Depending on how the motors are wired to the moto:bit, your micro:bot may drive in the opposite direction than what you would expect. The `set ____ motor invert to ____` provides the option to switch the motor wires in code without physically rewiring the motors to the moto:bit. Set the `left` or `right` motor that you would like to invert. If you want the motor to drive in the opposite direction relative to your setup, select either `true` or `false`.

On Button Pressed

The `on button __ pressed` code block is an event function. Any code blocks that are placed inside of the block will only execute when that event happens. In this block, when the A button is pressed on your micro:bit your robot will start moving. When it gets to the end of the program it will stop again until you press the A button again.

Turn Motors

The `turn motors __` block gives you the ability to turn the motors `ON` or `OFF` in software. They are naturally `OFF`, so anytime you want to drive motors, you need to set them to `ON`. A good practice is to turn them `OFF` when your robot completes its task or should not be moving for long periods of time, this saves batteries and wasted

power consumption.

Move Motors At

The `move ____ motor _____ at _ %` block is the basis of the moto:bit software package. You use this block by selecting the values in the drop down menu:

- which motor you want to control (`RIGHT` or `LEFT`)
- its direction (`FORWARD` or `REVERSE`)
- the throttle / power percentage you would like the motor to spin at (`0 - 100` %)

For your bot to move forward you need to have both motors drive in the same direction. To turn, or pivot, you set the motor directions in an opposite configuration.

Pause

The `pause (ms) ____` block is like a code stop sign. It tells the micro:bit to wait for a given amount of time in milliseconds. While it is waiting, whatever you told before the `pause` will keep happening. So, if you want your robot to drive forward for 1 second, you set the robot's motors to drive forward and then pause for 1000 milliseconds and then have the motors do something else, like stop.

What You Should See

Once your script uploads to your micro:bit, make sure the motors switch is changed from "STOP MOTORS" to "RUN MOTORS." Press the A button on your micro:bit and your robot will drive forward for 1 second, pivot, and the drive forward for another second before stopping. Pressing the button again will have the robot repeat the sequence.
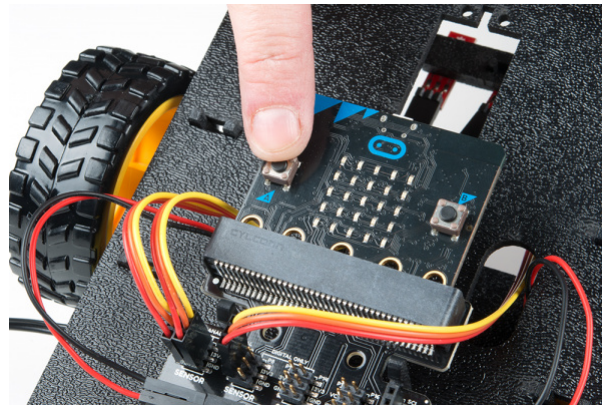
Here is a demo of the micro:bot driving.

micro:bot kit: Exp1

But wait? The micro:bot appears to be driving in the **reverse** relative to the front of the robot even though we told the robot to move `forward` in the code?!? There are a few reasons why this may be happening. Depending on how the motors were manufactured, they wires may be switched. Or the wires were connected incorrectly.

One method is to use the `set ____ motor invert to ____` block for each motor without rewiring the motors. Simply invert the motors setting each to `true`. The following experiments will invert the motor wire connection relative to how we assembled the robot earlier.
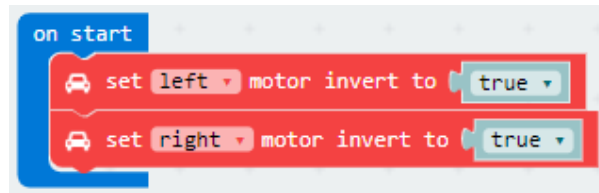
The other method is to physically rewire the motor wires. Simply flip the wire connections for each channel in order for the robot to move forward. Connect the "**LEFT MOTOR**" channel's red wire to the silkscreen labeled as "**BLACK**" and black wire wire to the silkscreen labeled as "**RED**". Then flip the wires for the "**RIGHT MOTOR**" channel with the right motor. The wiring may look similar to the image shown below on the right. If you flip the wires after this point, make sure to `set left motor invert to ____` and `set right motor invert to ____` back to `false` for future experiments.

Before Flipping the Wires	After Flipping Wires
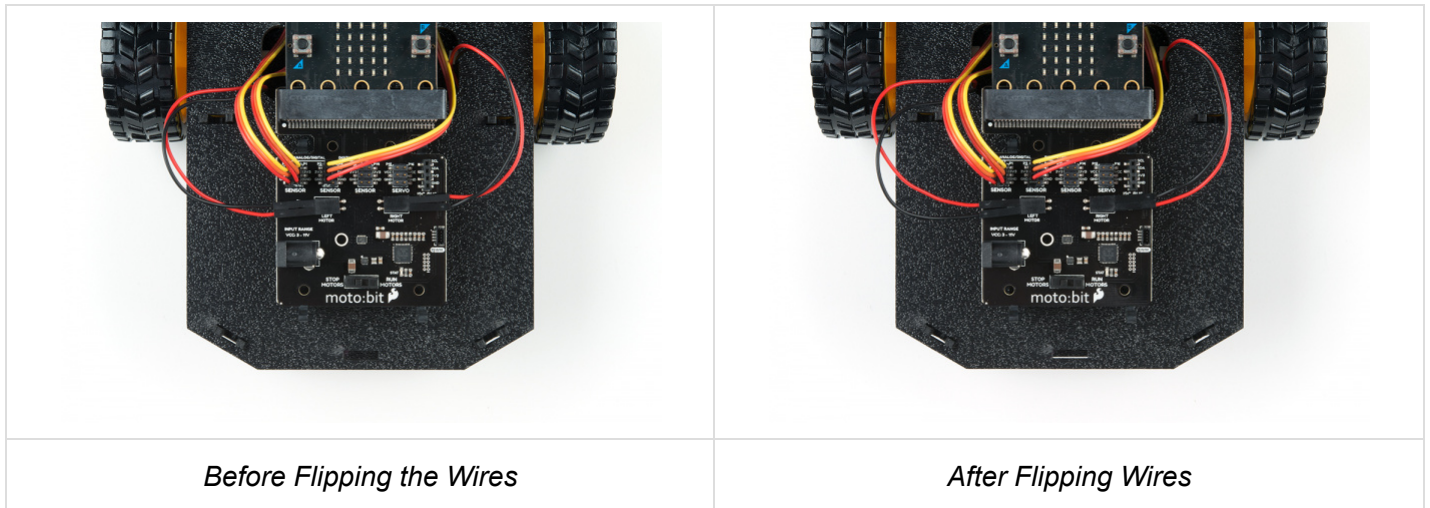
Go Further: Now that you have a moving robot, can you write a program that would tell your robot to drive in a square? How about a star? What about having it dance?

Troubleshooting

- **moto:bit package is not showing up** - Try installing it again from the add package option in MakeCode

- **Micro:Bit Not Showing Up On My Machine** - Try unplugging the USB cable and plugging it back in. Also, be sure that you have the cable inserted all the way into your micro:bit

- **Robot Not Moving** - Make sure your motors are hooked up correctly in the motor ports and your motor power switch is set to "RUN Motors", the battery pack is plugged in, and you have fresh batteries.

- **micro:Bot Not Driving in a Straight Line** - Try adjusting left or right motor's % to calibrate.

- **Moving in Reverse?!** - There are two options if you notice the micro:bot moving in the wrong direction. As explained above, you can flip the wiring of your motors or use the `set ____ motor invert to ____` block in your `on start` block to change what is forward vs. reverse.

Experiment 2: Staying in a Box

Introduction

Robots are smart! They are smart because they use sensors to detect what the world around them and then respond to those conditions. One of the conditions our robot can respond to is the darkness of the surface that it is driving on. It can detect lines, a gradient of darkness or a blacked out area. In this experiment you will program your robot to stay inside of a box that you have drawn on a surface.

Parts Needed

You will need the following parts:

- **1x** micro:bit board *(Not Included with Kit)*
- **1x** Micro-B USB Cable *(Not Included with Kit)*
- **1x** moto:bit Carrier Board
- **2x** Wheels
- **1x** Assembled Shadow Chassis
- **2x** Hobby Gear Motors
- **1x** 4xAA Battery Holder
- **4x** AA Batteries *(Not Included with Kit)*
- **3x** Analog Line Following Sensors
- **3x** 3-Pin Jumper Wires

Didn't get the kit? Have no fear! Here are the parts you will need to complete this experiment. You may not need everything though depending on what you have. Add it to your cart, read through the guide, and adjust the cart as necessary.

Hobby Gearmotor - 140 RPM (Pair)
◉ ROB-13302

USB micro-B Cable - 6 Foot
◉ CAB-10215

micro:bit Board

◉ DEV-14208

SparkFun moto:bit

○ DEV-14213

Wheel - 65mm (Rubber Tire, Pair)

◉ ROB-13259

Battery Holder - 4xAA to Barrel Jack Connector

◉ PRT-09835

SparkFun RedBot Sensor - Line Follower

◉ SEN-11769

Shadow Chassis

◉ ROB-13301

Panasonic Alkaline Battery - AA

◉ PRT-15201

Jumper Wire - 0.1", 3-pin, 6"

◉ PRT-10368

Suggested Reading

Before continuing on with this experiment, we recommend you be familiar with the concepts in the following tutorial:

Getting Started with the micro:bit
The BBC micro:bit is a compact, powerful programming tool that requires no software installation. Read on to learn how to use it YOUR way!

Introduction to the Line Sensors

The line follower sensor is an add-on for your shadow chassis that gives your robot the ability to detect lines or nearby objects. The sensor works by detecting reflected light coming from its own infrared LED. By measuring the amount of reflected infrared light, it can detect transitions from light to dark (lines) or even objects directly in front of it.
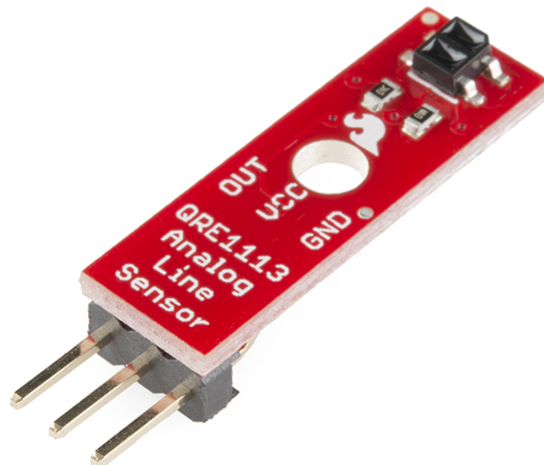
The sensor has a 3-pin header which connects directly to the moto:bit carrier board via female to female jumper wires. A mounting hole lets you easily connect one or more of these to the front or back of your robot chassis.

The IR Reflectance sensors work best when they are close to the surface below the micro:bot. The sensor should be about 1/8" above the table. This is an optimal distance for the IR transmitter to illuminate the surface below and measure the reflected light. *(Note: Human eyes are non sensitive to IR light, but if you use a CCD camera -- like*

the one in your phone -- you can see a small light shining out of the front element).
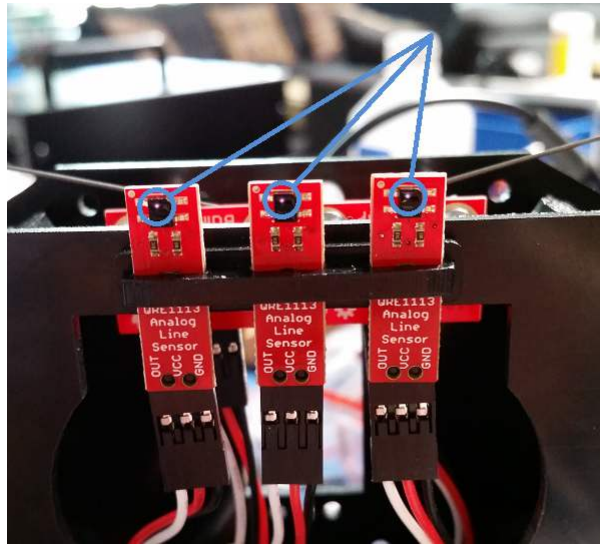
Here, you can see a faint pink glow from the IR LED. This is picked up on most digital cameras and cell phones. Note that the cable wires for the micro:bot are different as opposed to the ones used in this image.

Hardware Hookup

Like the motors, you should have already hooked up the line sensors during the assembly portion of this guide. You can go there now for the full assembly instructions. Double check to make sure that the wires are hooked up to your line sensors correctly!
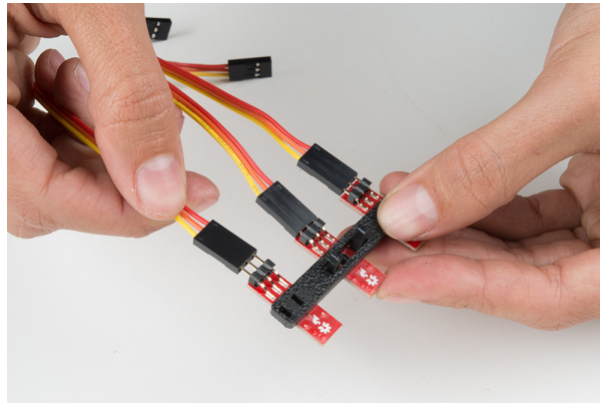
The line sensors hookup to your moto:bit via female / female jumper wires that snake through the chassis of your robot up to the moto:bit. The sensors hookup to the moto:bit in the following order:

- LEFT => `P0`
- CENTER => `P1`
- RIGHT => `P2`

Double check to make sure they are hooked up correctly and in the proper orientation:

Running Your Script

Be sure to add the moto:bit package as instructed in the Installing the moto:bit Package in MakeCode section of this tutorial.

Now, you can either download the following example script below and drag and drop it onto your micro:bit, or use it as an example and build it from scratch in MakeCode.

> **Calibration** is very important with the line sensors to work accurately. Your environment will greatly affect the P1 analog readings and thresholds for the surface value so you might have to customize these numbers to suit your application.

▶ Simulator   🖴 Blocks   {} JavaScript   ☑ Edit

```
on start
    serial redirect to USB
    set  left ▽  motor invert to  ‹true ▽›
    set  right ▽  motor invert to  ‹true ▽›
    set  black_line ▽  to  ‹analog read pin  P1 ▽›
    serial write string  " black_line_(i.e._Baseline_Value_of_Line_Following_Se
    serial write line  black_line ▽
```

Code to Note

Let's take a look at the code and what to expect.
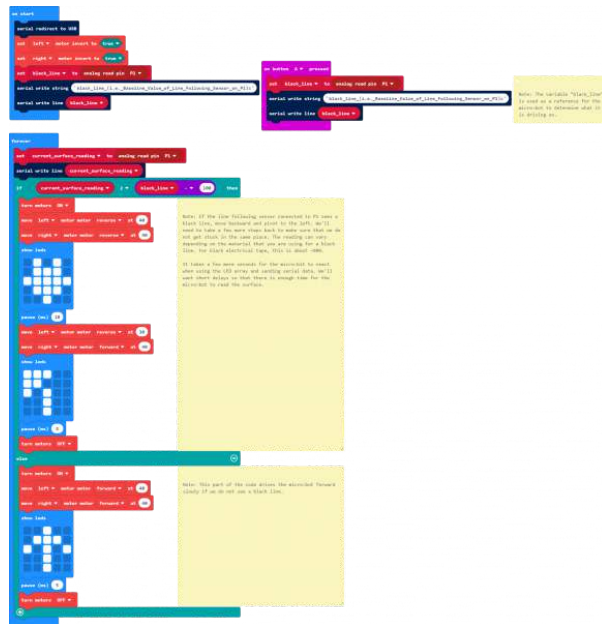
Click on image for a closer view.

Initialize Serial Output

When the code first starts, we initialize a serial output on the micro:bit to send serial to the USB. Using the `serial redirect to USB` code block defaults the **115200** baud. This is useful whenever we need to inspect the sensor readings and interpret the values in a serial terminal.

Set Motors

We will need to set the motors depending on how the motors are wired to the moto:bit. In this case, we will be setting both motors to `true` to be consistent with how we assembled the robot earlier.

Creating a Variable

In Microsoft MakeCode, you need to be able to create variables to store information for your robot to compare against, or to just remember. You can use the built-in variables under the Variables drawer as shown below.
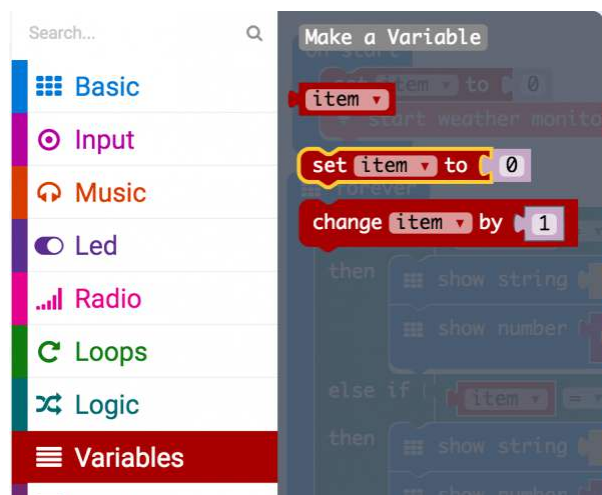
But, you can also make your own custom variables. To do this click on the Variables drawer and select Make New Variable. You can then name it and use it in your program. In this case, we are going to create two new variables. One to have the micro:bit remember what our black line looks like (i.e. `black_line`) and another for the current

sensor reading (`current_surface_reading`).
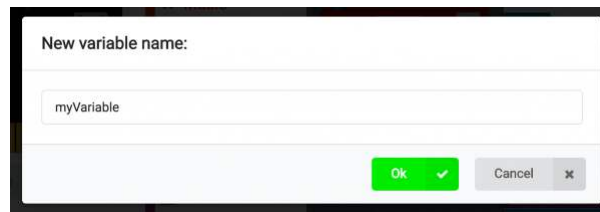
Set To

To store a value or piece of information in a variable you use the `set _____ to` block. This allows you to select a variable and set it to a value or block of your choice. In this case, the line following sensor value connected to pin `P1` . We do this once in the `on start` block to get a base reading from the sensor so that our micro:bit remembers the value of the black line. This calibrates the sensor for the micro:bot. Pressing button A while the program is running will also save the line sensor reading again.

Analog Read Pin

To get that base reading from the line sensor we use the `analog read pin __` block to get an analog value (`0` - `1023`) from the line sensor. `1023` is totally black and `0` is totally saturated white. Your surface will be something probably in between.

If Condition Statements

To compare the current reading of the line sensor to its known baseline value that was captured and stored in the `on start` block, we use an `if` statement block. The `if` block accepts a logical statement. If that statement (`current_surface_reading ≥ black_line - 100`) is true, then the `if` block runs then "then" section of code. If that statement is false, then the "else" section of the block is run.

You may be asking yourself why subtract `100` from the `surface` value. Well, that number is a sensitivity value. The sensor reading fluctuates as the robot moves around and when the motors are running. By subtracting `100` from the original surface variable, we give the change in lighting a "wiggle room" or tolerance of `100` . If you want your sensor to be more sensitive, you would make the number smaller. If you need it less sensitive, make it bigger.

To debug, we'll send serial to the USB before the condition statement and animate the LEDs of the micro:bit within each condition statement `show icon _____` (or `show leds _____`) block to help visualize when the micro:bot moves forward, back, or to the left.

What You Should See

Serial Output

Heads up! To view the serial output using the Chrome browser and the MakeCode's serial plotter and terminal, the micro:bit will need to have **0249, 0250 or higher**. To update, make sure to follow these instructions from micro:bit's support to display live serial data MakeCode console.

MICRO:BIT FIRMWARE UPDATE

You can pair, upload code to the micro:bit, and view the output on the MakeCode console without having to drag and drop the file to the micro:bit after updating the firmware. For more information about using the WebUSB feature on MakeCode, make sure to check out the instructions provided by micro:bit support.

For the scope of this tutorial, we'll use the MakeCode console to output the serial output. If you have paired the micro:bit to your computer and used the one-click download feature for MakeCode, a "**Show console Device**" button should appear in MakeCode. Click on the button to view the serial output. A graph and serial monitor will appear over the MakeCode code editor. You may notice that there is no data graphed and a number on the bottom left. The serial monitor will also display a number with a counter. The counter indicates how many times our output the same value. This is due to the graph automatically resizing. Try moving the line following sensor around the surface of a table. Keep in mind that the micro:bit in the simulator does not reflect the LEDs on the physical micro:bit in real-time.

Heads up! Each time you disconnect the micro:bit from your computer, you will need to re-connect the micro:bit by pairing it to the web browser. The "**Show console Display**" button may disappear. After pairing the micro:bit again and using the one-click Download feature, the button should re-appear.

You will notice that the data will start jumping around as the line following sensor on P1 reads the surface. Below is a snapshot of the line following sensor on a white table. The surface that the sensor was reading was not perfectly white and caused the sensor readings to jump around **~693 to ~721**. You may see a different output depending on the line following sensor, surface, and distance between the sensor and surface.

To better understand what is going on when the line following sensor sees a dark black line, we'll place different surfaces under the sensor. We'll use the following three surfaces to illustrate what is happening in the graph:

- white table
- brown cardboard
- black electrical tape

White Table under P1	Brown Cardboard under P1	Blac

The MakeCode serial plotter automatically zooms in/out so it may be hard to view the data at first. Placing a different surface under the sensor with the micro:bot staying still makes it easy to see the output once the console zooms out. Here's the raw output after placing the table, cardboard, and electrical tape under the sensor for a few seconds.

Let's look at the same graph but highlighted for the different surfaces. The output highlighted in blue was the surface that we initially read (which was **~693 to ~721**). The small changes in the readings seem insignificant when comparing the output to other surfaces. Placing a small piece of brown cardboard under the sensor will cause the sensor readings to drop a certain range. The surface of the cardboard was a bit higher than the surface of the table. In this case, it was about **~82 to ~85** as highlighted in green. Moving the sensor over a dark black line will cause the sensor readings to jump to a certain range again. In this case, the values were around **~943 to ~983** as highlighted in red. Feel free to take some notes down on paper to remember what your range is for different surfaces.

If you turn on the motor and have the micro:bot take a few steps across each surface, you'll notice that the output starts becoming noisy instead of being a steady output. This is due to the small differences across the surface and the IR light being reflected back while the micro:bot is moving. There also might be some noise coming from the

batteries whenever the motors pull power and sunlight coming from the windows.


If we inspect the output when the sensor is over the white table (highlighted in blue), the readings jump around **~632 to ~813**. This is a bit bigger than what we initially read. If we inspect the output when the sensor is over the brown cardboard surface (highlighted in green), the readings jump around between **~60 to ~650**. This is bigger than what we initially read. One reason for this is that the brown cardboard is not uniform throughout the material. Additionally, the distance between the line following sensor and cardboard became the same as when it was driving over the white table. If we inspect the output when the sensor is over black electrical tape (highlighted in red), the readings jump around **~932 to ~988**. This is a bit bigger than what we initially read with the white table but not as big as the brown cardboard surface.

Note: The distance between the line following sensor and the surface matters! Make sure that your surface is flat and the wheels are on the surface that it is traveling on. Any deformities on the surface can affect the line following sensor readings. Remember to have a distance of about 1/8" above the surface by having the wheels and nub caster over the material that it is traveling on.
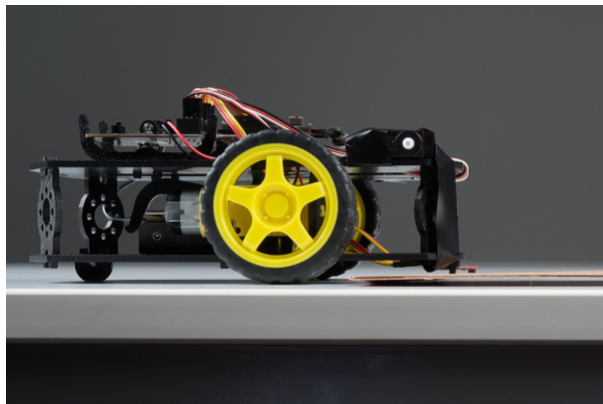
Distance Between Line Following Sensor and Surface Too Close
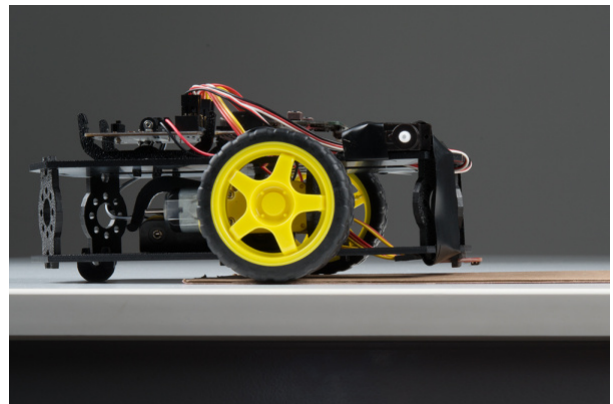
*Optimum Distance Between Line Following Sensor and Surface*

Now that we understand the output, we can head back into the code to adjust it as necessary where it says `black_line - 100` to distinguish a light surface (white table or brown cardboard) from a dark surface (i.e. black electrical tape). Since we know that the values for a dark black line will be anything above **~932**, we take that baseline value connected to P1 and subtract it by `100`. Subtracting it by a value of `100` gives the line following sensor some tolerances in case there is an error in reading a black line. As a result, anything above **~832** will be recognized as a black line. The image below highlights the area in red where it is `black_line - 100`. If the current reading is above the boundary (indicated by the red line), the micro:bot will recognize it as the black electrical tape. Keep in mind that the analog sensor is not able to go higher than `1023`, so anything between `~823 to ~1023` will be recognized as a dark line and does not need to be included in the code. Depending on your surfaces, you may need to adjust the value as necessary to change the sensitivity.
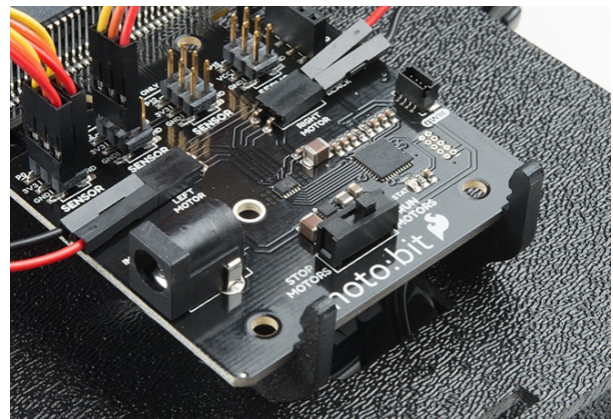

Roaming micro:bot

Once your code is loaded find a light surface, preferably white and add a line of black electrical tape to the surface. For simplicity, let's make the width of the line between 1.5 inches and 2.0 inches. Continue adding black electrical tape until you make a square. In fact, if you have a large enough space you can create a 3 foot square you will be perfect! Place the center line following sensor connected to P1 on the black line. Then press the RESET button on your micro:bit. The micro:bot will remember what a black line looks like. Make sure to change the motor switch from "STOP MOTORS" to "RUN MOTORS." If you ever need to re-calibrate the sensor while the code is running, simply place the line following sensor connected to P1 over the dark black line and press the A button.
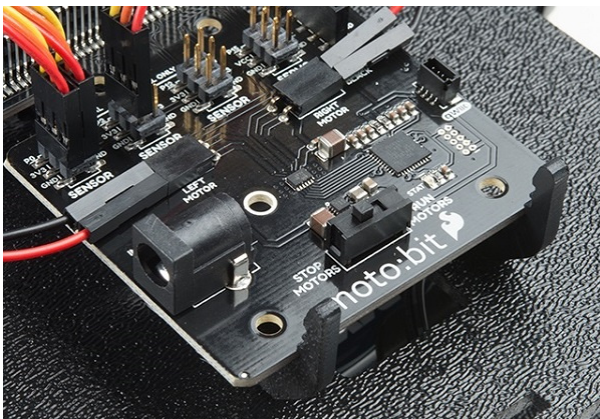
| *Switch Flipped to STOP MOTORS* | *Switch Flipped to RUN MOTORS* |

The micro:bot will try to move away from the black line by backing up and turn to the left a bit. Your robot will attempt to drive forward until the black tape is directly under the center line sensors. When that happens your robot will stop, reverse, pivot to the left a bit, and try to drive forward again before hitting a line once more. If it

works correctly and your robot is within the black square line, your robot should stay inside of the box forever!

micro:bot kit: Exp 2
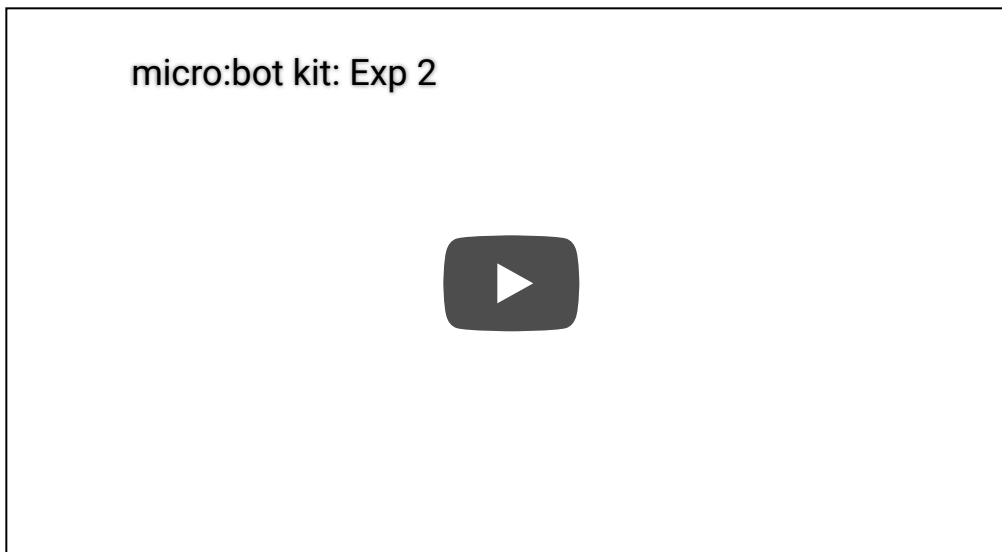
> **Go Further:** So, you have built a robot corral! There is a lot of resources being used causing the micro:bot to drive slow. Try removing the serial output, removing the `show LEDs` code blocks, adjusting the speed of the motors, and tweaking the `pause` blocks to see if you can get the micro:bot to move faster in the square. Or maybe try to use what you now know to build a maze for your robot to solve on its own! Better yet, try adjusting the logic used in the `if` statements to have the micro:bot driving on a black surface and contained within a white line.

Troubleshooting

- **Moto:Bit blocks not showing up** - Make sure you have a network connection and you have added the package to your makeCode environment.

- **Micro:Bit Not Showing Up On My Machine** - Try unplugging the USB cable and plugging it back in. Also, be sure that you have the cable inserted all the way into your micro:bit

- **Robot Not Moving** - Make sure your motors are hooked up correctly in the motor ports and your motor power switch is set to "RUN Motors", the battery pack is plugged in, and you have fresh batteries.

- **Robot Not Driving Forward** - If your motors are hooked up correctly and we have inverted both motors in the code, try hitting the reset button on the micro:bit. This can happen after uploading when initially powering the micro:bit up.

- **Moving in Reverse?!** - There are two options if you notice the micro:bot moving in the wrong direction. As explained above, you can flip the wiring of your motors or use the `set ____ motor invert to ____` block in your `on start` block to change what is forward vs. reverse.

- **Robot Drives Over the Black Line** - Your motors may still be running as the micro:bit is updating the LEDs pausing, or sending serial data to the USB. Make sure to have the micro:bot drive forward slowly (but not to slow where the motors will not move) before taking an additional sensor reading. You can also try to adjust the values or remove the blocks of code used for debugging. If the micro:bit drives to fast, it will not be able to see the width of the black line. The width of the line may be too small or the material for the black line

might not be dark enough for the micro:bot to see. Try increasing the width of the black line or changing the material (i.e. besides black electrical tape, try using a darker paint, marker, pencil, etc.) used for the black line.

- **Robot Doesn't Detect Line** - If changing the width of your line doesn't help, remember line sensor calibration settings can be very sensitive depending on your environment. Press the A button to calibrate when the robot is on the `black_line` or change the value being subtracted from the `black_line`.

Experiment 3: Following a Line

Introduction

OK, you have your robot staying inside of box drawn on the floor, but that still seems a little odd and random. You want your robot to go somewhere, do something and then keep going! In this experiment, you will elaborate what you learned from Experiment 2 to get your robot to follow a line.

Parts Needed

You will need the following parts:

- **1x** micro:bit board *(Not Included with Kit)*
- **1x** Micro-B USB cable *(Not Included with Kit)*
- **1x** moto:bit Carrier Board
- **2x** Wheels
- **1x** Assembled Shadow Chassis
- **2x** Hobby Gear Motors
- **1x** 4xAA Battery Holder
- **4x** AA Batteries *(Not Included with Kit)*
- **3x** Analog Line Following Sensors
- **3x** 3-Pin Jumper Wires

Didn't get the kit? Have no fear! Here are the parts you will need to complete this experiment. You may not need everything though depending on what you have. Add it to your cart, read through the guide, and adjust the cart as necessary.

Hobby Gearmotor - 140 RPM (Pair)
◉ ROB-13302

USB micro-B Cable - 6 Foot
◉ CAB-10215

micro:bit Board

◉ DEV-14208

SparkFun moto:bit

◯ DEV-14213

Wheel - 65mm (Rubber Tire, Pair)

◉ ROB-13259

Battery Holder - 4xAA to Barrel Jack Connector

◉ PRT-09835

SparkFun RedBot Sensor - Line Follower

◉ SEN-11769

Shadow Chassis

◉ ROB-13301

Panasonic Alkaline Battery - AA

◉ PRT-15201

Jumper Wire - 0.1", 3-pin, 6"

◉ PRT-10368

Suggested Reading

Getting Started with the micro:bit
The BBC micro:bit is a compact, powerful programming
tool that requires no software installation. Read on to
learn how to use it YOUR way!

Introduction to Using Multiple Line Sensors

In the previous experiment, you used a single line sensor (the middle sensor) to detect the line on the floor. That is great for staying inside of a line, but now, you need to follow a line. That is where the other two line sensors come in.

Essentially, you want the center sensor to detect the line, but not the other two, meaning that the robot is centered on the line. If one of the side sensors detect a line it means that you are veering to one side or another and your robot should correct itself. We will use the information from multiple sensors combined with an `if/else` statement block to build a decision tree for your robot to follow. For simplicity, we will start with using just two of the line sensors (the left and right sensors) to follow a dark black line. As you add more line following sensors to a robot, the code can get complex. However, the robot will able to follow a line better.

P0 and P2 Reading a Black Line

Hardware Hookup

> **Note:** If you already hooked up your sensors in Experiment 2, please skip this section.

Like the motors, you should have already hooked up the line sensors during the assembly portion of this guide. You can go there now for the full assembly instructions. Double check to make sure that the wires are hooked up to your line sensors correctly!
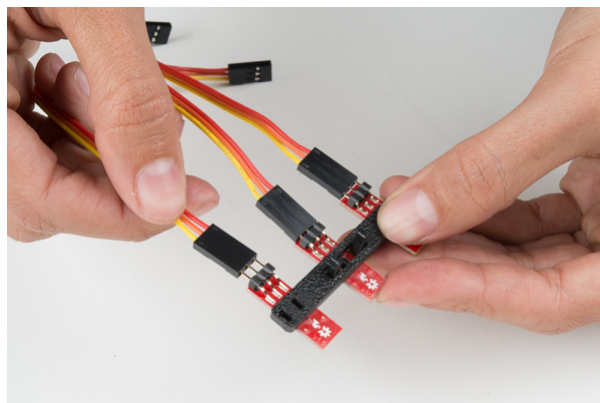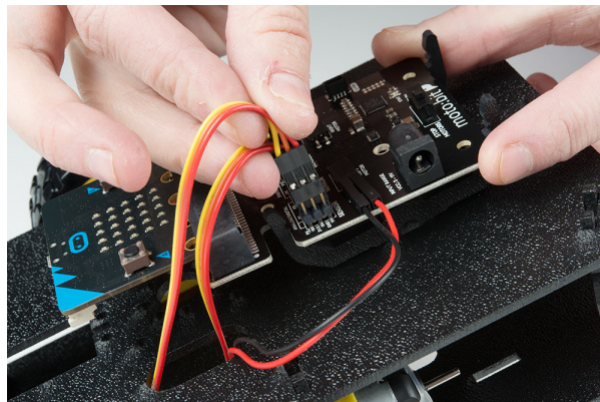
The line sensors hookup to your moto:bit via female / female jumper wires that snake through the chassis of your robot up to the moto:bit. The sensors hookup to the moto:bit in the following order:

- LEFT => P0
- CENTER => P1
- RIGHT => P2

Double check to make sure they are hooked up correctly and in the proper orientation
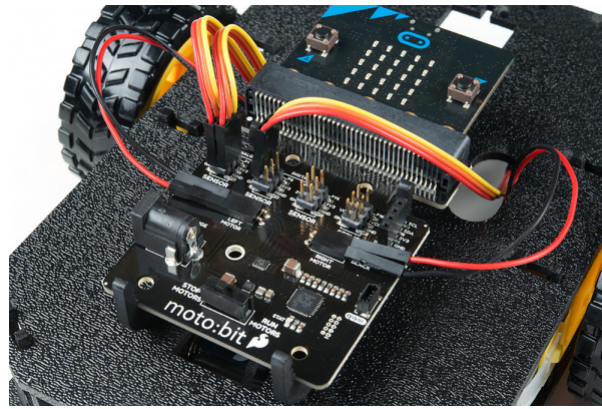
Experiment 3a -- Simple IR Sensor Reading

Let's break up this experiment into two parts. First, we will have the micro:bot follow a straight dark, black line. Then we will have a slightly more complex path to try to have the micro:bot drive around a path with a zigzagged pattern.

Running Your Script

Be sure to add the moto:bit package as instructed in the Installing the moto:bit Package in MakeCode section of this tutorial.

Now, you can either download the following example script below and drag and drop it onto your micro:bit, or use it as an example and build it from scratch in MakeCode.

> **Calibration** is very important with the line sensors to work accurately. Your environment will greatly affect the P0 and P2 analog readings and thresholds for the surface values so you might have to customize these numbers to suit your application.

▶ Simulator   🧩 Blocks   {} JavaScript   ☑ Edit
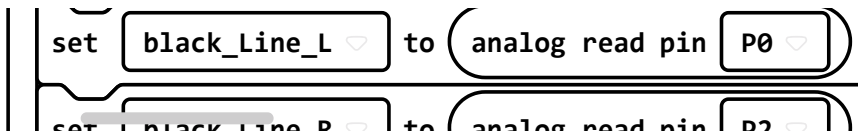
```
on start
    serial redirect to USB
    set  left ▽  motor invert to  true ▽
    set  right ▽  motor invert to  true ▽
    turn motors  OFF ▽
```
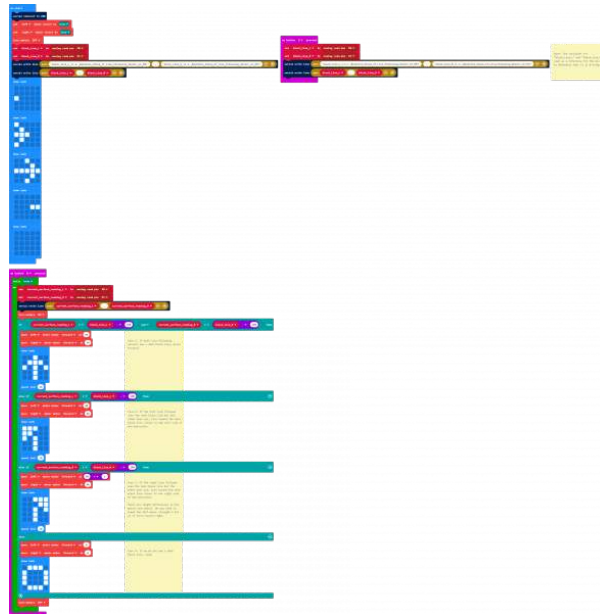
Code to Note

Let's take a look at the code and what to expect.

Click on image for a closer view

black_Line_L and black_Line_R

Like in the previous experiment, you need to set a baseline value for the surface that your robot is driving on. This is actually called a calibration value. We need to do this for two sensors now; left and right. We go through the same routine we did for the single sensor previously, but for the left and right sensors.

Comma Delimiter

For debugging, we set up the serial again. However, we will use the `join` block with a comma delimiter. This is useful for graphing more than one sensor readings.

On Button Press

As in the first experiment, we use the `On Button Press` block to start the program. Since we are using button A again to recalibrate the line following sensors whenever we need, we'll be using button B to start the program. This is so you can get a good base reading to calibrate your sensors without having to wrestle with a robot that is trying to move around. To remind us to press button B, the `on start` block displays a short animation once to point at button B. If the motor switch is flipped to the RUN MOTORS side, pressing button B will cause the micro:bot to start following a line.

While

The `While` block is a logic block that is similar to the `loop` block, but a bit smarter. The `While` block accepts a TRUE/FALSE statement, if that statement is true the block will loop whatever code is placed inside of it. If that value is false, the `While` block is simply skipped over. We hardcode a `true` value into the `While` block so that it

constantly loops when the B button is pressed. That way we have the benefits of both the event block and the `loop` block without needing complicated programming.
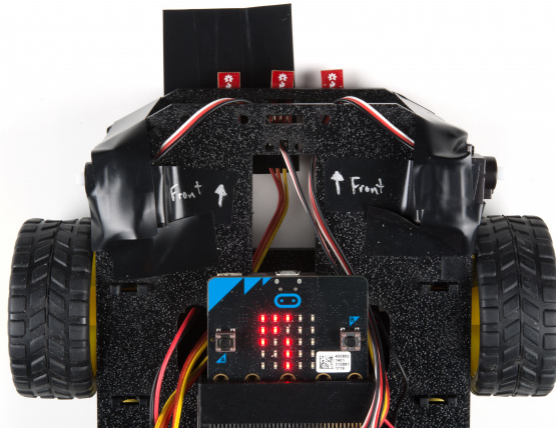
If Condition Statements

We'll use `if/else` statements once again to check the sensor readings and move the micro:bot:

- **Case 1**:
   - If both are reading the same, **move forward**.
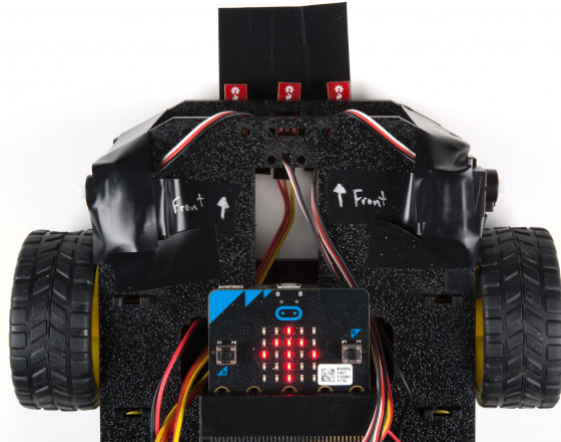
- **Case 2**:
    - If the left sensor sees a dark black line but the other does not, **move forward and a little to the left** toward the dark black line closer to the left side of the micro:bot.

- **Case 3**:
    - If the right sensor sees a dark black line but the other does not, **move forward and a little to the right** toward the dark black line closer to the right side of the micro:bot. There are slight differences in the motors and wheels so we tweak the left motor strength a bit so that the micro:bot actually turns toward the right (`50 + 5`).
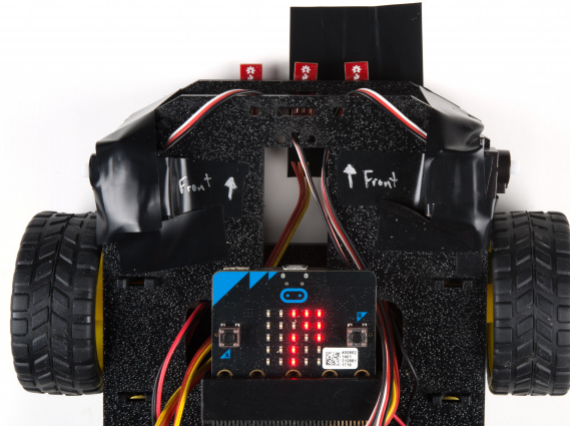
- **Case 4**:
    - If we do not see a dark black line, **stop**.
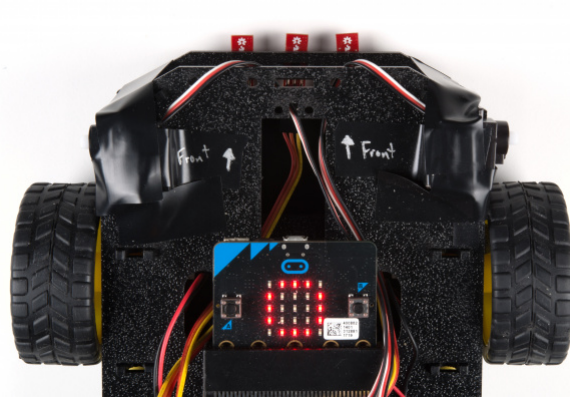
What You Should See

Serial Output

> **Heads up!** To view the serial output using the Chrome browser and the MakeCode's serial plotter and terminal, the micro:bit will need to have **0249, 0250 or higher**. To update, make sure to follow these instructions from micro:bit's support to display live serial data MakeCode console.
>
> <div align="center">
>
> **MICRO:BIT FIRMWARE UPDATE**
>
> </div>
>
> You can pair, upload code to the micro:bit, and view the output on the MakeCode console without having to drag and drop the file to the micro:bit after updating the firmware. For more information about using the WebUSB feature on MakeCode, make sure to check out the instructions provided by micro:bit support.
>
> <div align="center">
>
> **DISPLAYING REAL TIME SERIAL DATA IN THE MAKECODE CONSOLE**
>
> </div>
>
> Otherwise, you can use your favorite serial terminal or program to real time, serial data plotter to graph the output.

We'll use the MakeCode console to output the serial output again. If you have paired the micro:bit to your computer and used the one-click download feature for MakeCode, a "**Show console Device**" button should appear in MakeCode. Click on the button to view the serial output.

You will notice that the data will start jumping around as the line following sensors on P0 and P2 read the surface. Below is a snapshot of the line following sensors on a light surface. Notice that the output of both sensors were not the same. This is due to the slight differences in the line following sensors. You may see a different output with your sensors.


Moving the micro:bot over a piece of brown cardboard and black electrical tape, you will notice that the sensor readings move at the same time when the graph zooms out.

| *White Table under P0 and P2* | *Brown Cardboard under P0 and P2* | *Black El* |
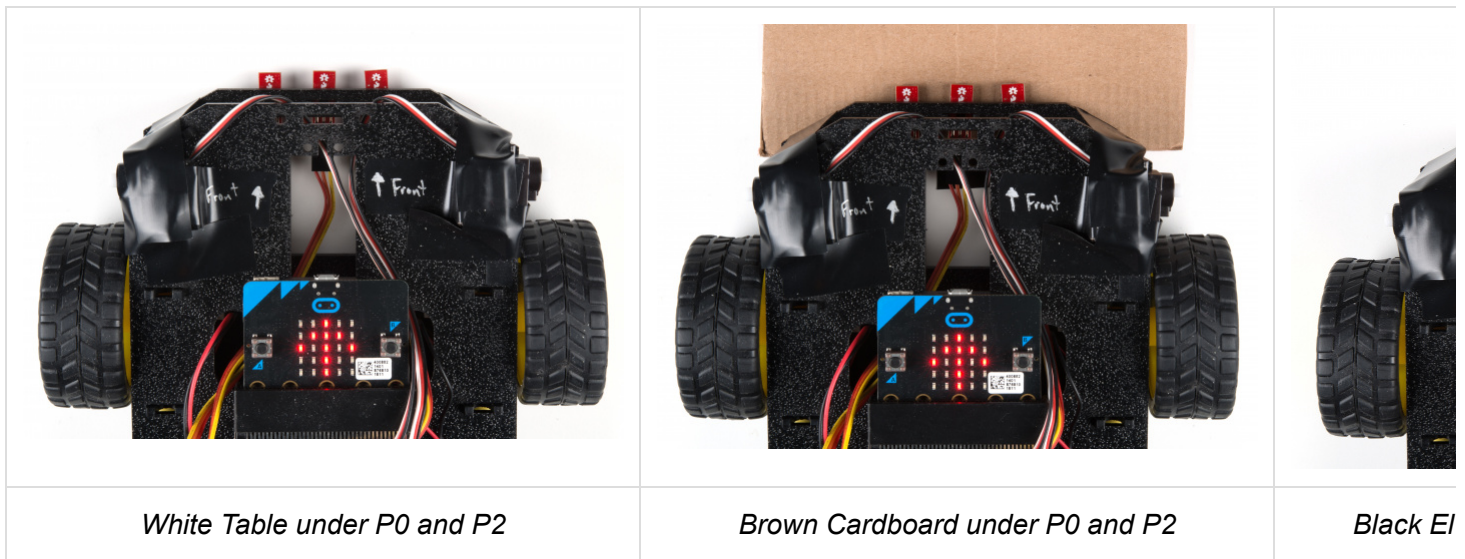
For this reading, we moved the micro:bot further into the cardboard so that the distance between the line following sensors and cardboard box were the same.


Let's look at the same graph but highlighted for the different surfaces. Overall, the output is similar to what we saw in experiment 2 when the motors are turned off.
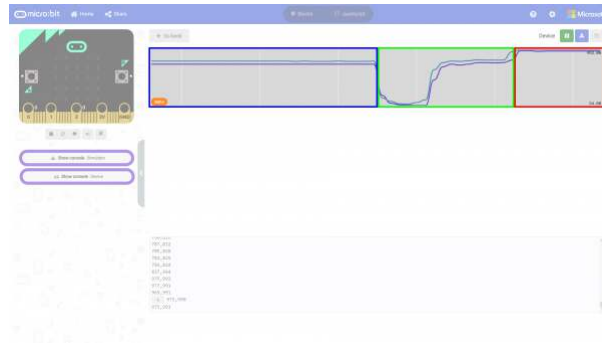
The image below highlights the area where it is above `black_line_L - 100` and `black_line_R - 100`. Note that we need to do this for the left and right line following sensor due to the slight differences. As a result, anything above **~875** for the left line following sensor (as indicated by the red line) is probably a dark surface. Additionally, anything above **~892** for the right line following sensor (as indicated by the black line) is probably a dark surface.
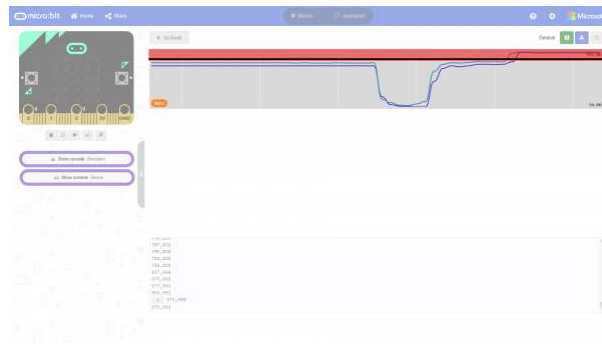

As we turn on the motors for the micro:bot and view the data, the output will become noisy instead of being a steady output. You'll also see the output become slightly bigger when ever the motors are turned on whenever it tries to follow a dark black line. The image below highlights the area when both sensors see a dark black line. When this happens, the micro:bot will drive forward.
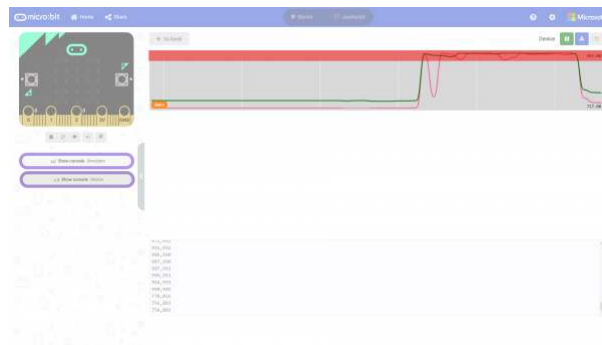
The image below shows when the left line following sensor (when it is above `black_line_L - 100`) sees a dark black line. The motors turn on and tries to move toward the line on the left. Note that the right line following sensor stays around the same value but becomes noisy.
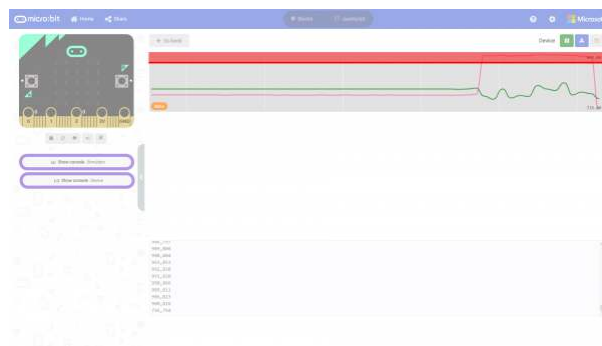
The image below shows when the right line following sensor (when it is above `black_line_R - 100`) sees a dark black line. The motors turn on and tries to move toward the line on the right. Note that the left line following sensor stays around the same value but becomes noisy.
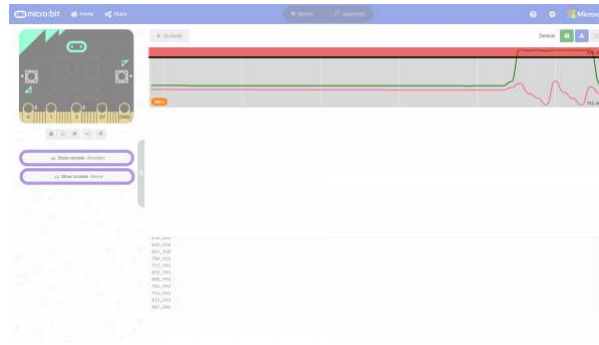
Straight Line Following micro:bot

Once you have loaded your script, place your robot on a dark black line on a light / white background. Make sure you have it centered with the line just underneath the center line sensor. Make sure the motor switch is changed from "STOP MOTORS" to "RUN MOTORS" and press the B button to start.

Stop Motors	*Run Motors*
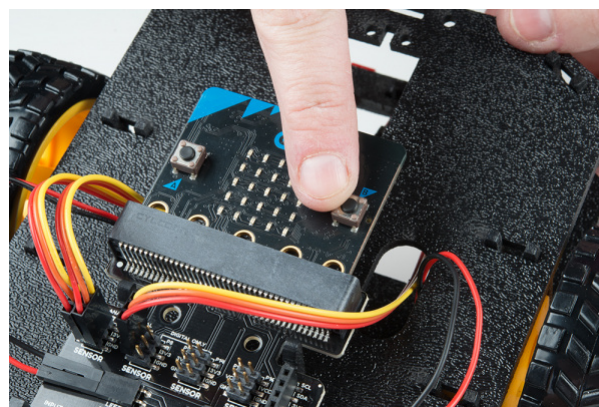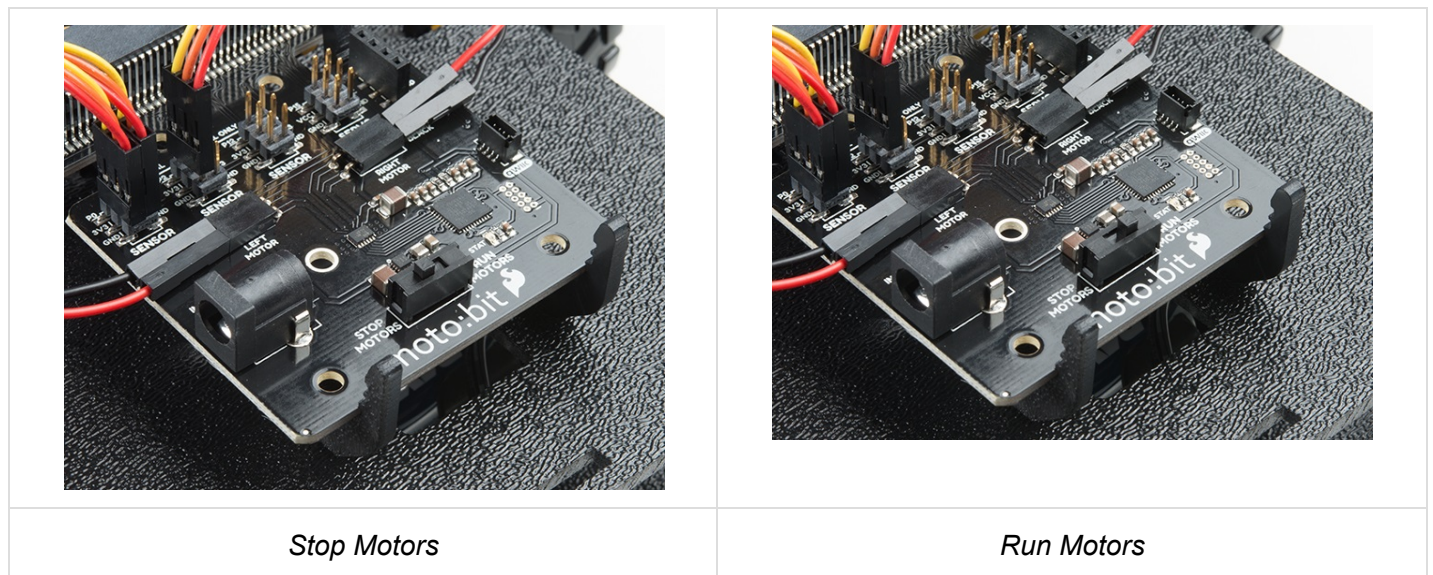
Your robot should drive forward until one of the side line sensors detects the dark black line and then it will turn in that direction to correct itself. Depending on your line shape and thickness your robot may "waddle" more or less. If you have the motors running too fast, it will continue moving until it sees a light surface. Make sure to have the robot slowly move forward or it may fall of the edge of a table!

b   k

micro:bot kit: Exp3

Experiment 3b -- Following a Course with a Zigzags

Now that the micro:bot can follow a straight line, let's try to have the micro:bot follow simple polygon with zigzags.

Running Your Script

Be sure to add the moto:bit package as instructed in the Installing the moto:bit Package in MakeCode section of this tutorial.

Now, you can either download the following example script below and drag and drop it onto your micro:bit, or use it as an example and build it from scratch in MakeCode.

> **Calibration** is very important with the line sensors to work accurately. Your environment will greatly affect the P0 and P2 analog readings and thresholds for the surface values so you might have to customize these numbers to suit your application.

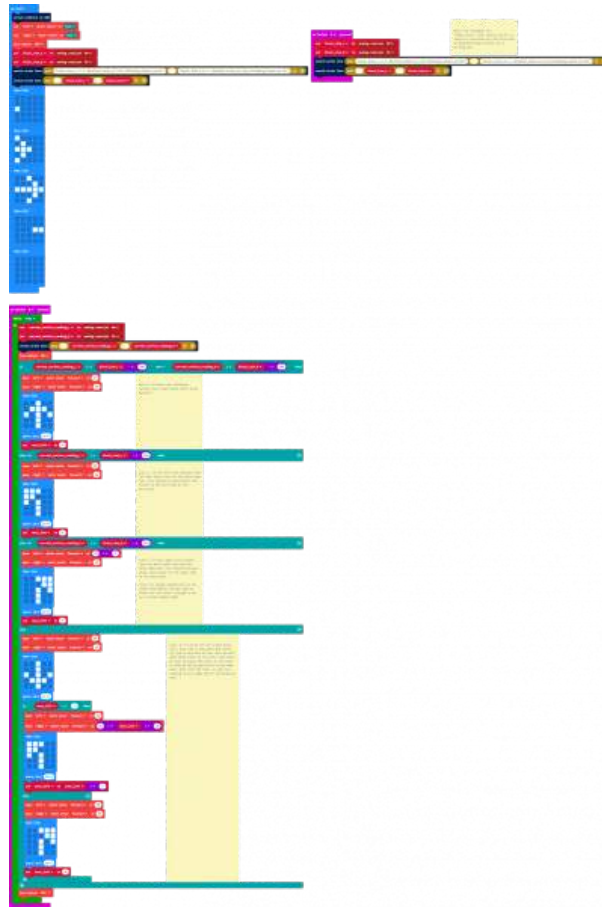▶ Simulator   📷 Blocks   {} JavaScript   ↗ Edit

```
on start
  serial redirect to USB
  set  left ▽  motor invert to  ⟨true ▽⟩
  set  right ▽  motor invert to  ⟨true ▽⟩
  turn motors  OFF ▽
  set  black_Line_L ▽  to  ( analog read pin  P0 ▽ )
  set  black_Line_R ▽  to  ( analog read pin  P2 ▽ )
```

Code to Note

Let's take a look at the code and what to expect.


Click on image for a closer view

Modified If Condition Statement

Most of the code used for experiment 3B is pretty much the same when following a black line. You should see the same condition statements for case 1 through case 3 when using P0 and P2 as the micro:bot attempts to follow a dark black line.

|		
Case 1 - Moving Forward	*Case 2 - Moving Forward and Left*	*Case 3*
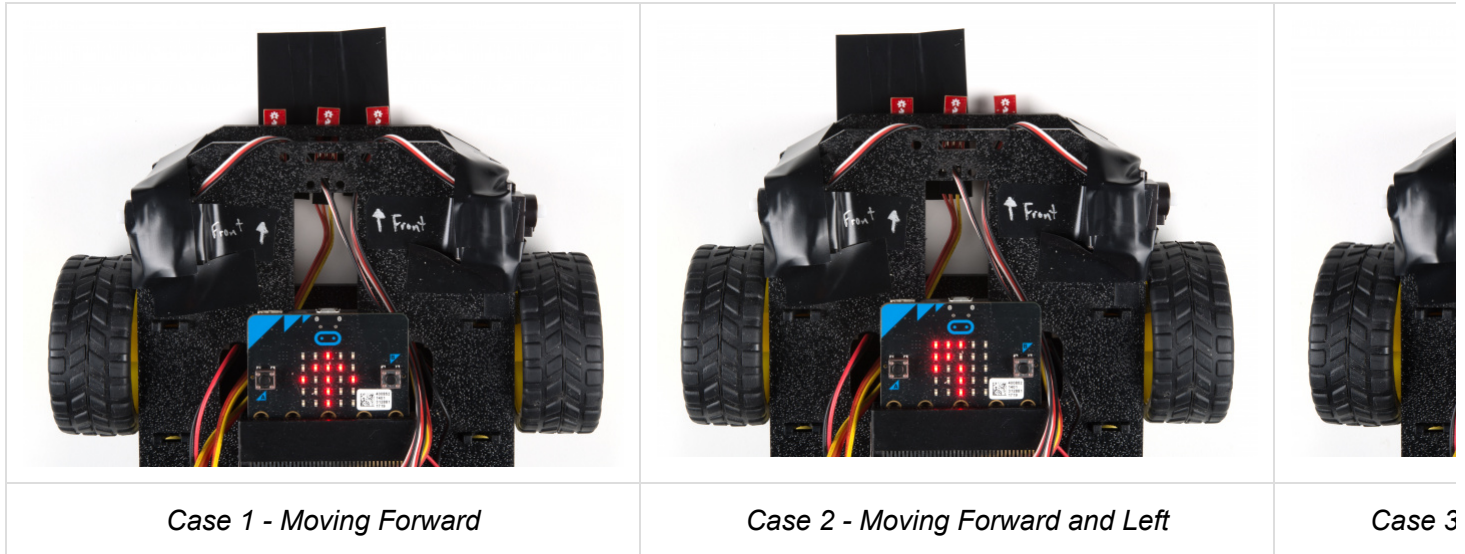
However, case 4 will be adjusted to keep searching for a path. If we do not see a dark black line, we will take a step back and search three times to the left. The variable `move_left` will keep track of how many times we have looked to the left. Each time we search to the left, we will increase the power of the motor so that we do not get stuck in the same spot. After the 3rd time, we will try looking to the right before starting all over again.
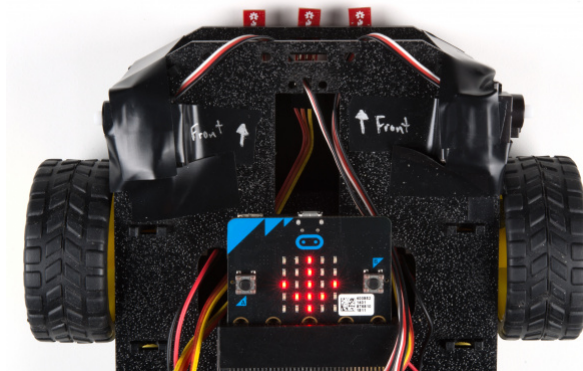

We will only want to use the `move_left` variable in case 4. We'll reset the variable whenever we go into the other cases.

What You Should See

The serial output will be the same if you open the MakeCode console. If you need, you can open it back up to view the output. If you have not already, make a track on a flat surface with varying edges. Make sure that there is enough of material adjacent to the track. If the wheels or nub caster run off the track, it may have problems trying to get back on the surface. We used a cardboard surface with black electrical tape for this experiment. Additional cardboard was attached to prevent the micro:bot from getting stuck.
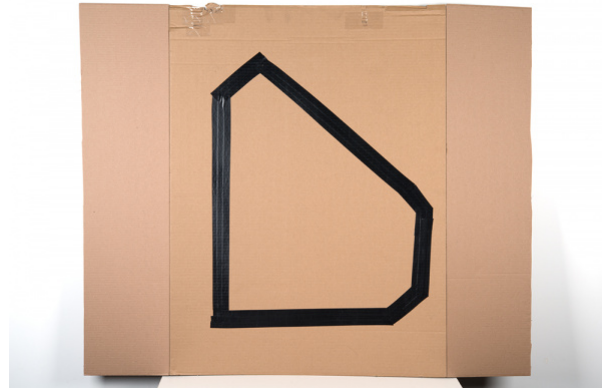
This example works well to follow small changes in the line for turns but will also adjust for sharp turns. You'll follow the same steps to get the robot to start following a line. The only difference is that the robot will follow a zigzagged course in this example. By default, the micro:bot will attempt to correct itself to eventually follow a track in a counter-clockwise motion. Keep in mind that there are limitations in following a dark black line when only using only two line following sensors. Additionally, the surface was not completely flat. As you can see from the GIF below that the floor was rocking as the micro:bot was following the line. Adding more line following sensors can help the micro:bot follow a line better and keep it from driving off the track.
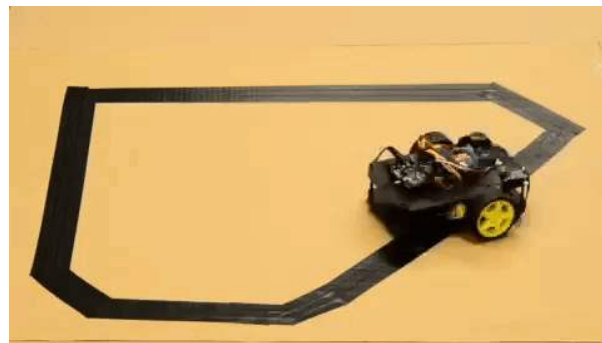
Go Further: Your robot can go where you tell it by following a line. We've just scratched the surface for line following problems. Here are some more challenges to explore and experiment:

- Try adding more code to utilize the center line following sensor.
- Instead of having the robot follow a counter-clockwise motion, try have the robot move in a clockwise motion.
- Adjust the code to follow a dark black line for a complex maze
- Adjust the motor speed and remove the LED array to see how fast your line following robot can move along the line without veering off the course.
- Use the built-in accelerometer to increase the motor speed whenever the micro:bot is on a slanted surface so that the robot can navigate rough terrain.
- Add an additional mode to invert the condition statements to follow a light white line against a dark surface instead of a dark black line against a light surface.
- Try adjusting the width of your track to see if the robot can follow a more narrow line.

Troubleshooting

- **Robot Drives In a Circle** - Double check your wiring and your code, your robot is not seeing the line. Also, double check that there isn't something like dust blocking your sensor.

- **Robot Isn't Moving** - Pesky motor switch! Make sure that is set to "run" and you have fresh batteries.

- **Robot Not Driving Forward** - If your motors are hooked up correctly and we have inverted both motors in the code, try hitting the reset button on the micro:bit. This can happen after uploading when initially powering the micro:bit up.

- **Moving in Reverse?!** - There are two options if you notice the micro:bot moving in the wrong direction. As explained above, you can flip the wiring of your motors or use the `set ____ motor invert to ____` block in your `on start` block to change what is forward vs. reverse.

- **Robot Waddles a Lot!** - Change the width of your line.

- **Robot Doesn't Detect Line** - If changing the width of your line doesn't help, remember line sensor calibration settings can be very sensitive depending on your environment. Also, the code calibrates when the micro:bot starts up. Make sure to have the sensors over a black line.

Experiment 4: Using the Accelerometer

Introduction

Now, what happens when something bumps into your robot? It just stays there and take it from this bully? No! It should move out of the way, or do something. In this experiment you will use the accelerometer on the micro:bit to trigger the robot to move out of the way. If you flip the robot on its back or stand it on end it will stop driving (play dead)!

Parts Needed

You will need the following parts:

- **1x** micro:bit board *(Not Included with Kit)*
- **1x** Micro-B USB Cable *(Not Included with Kit)*
- **1x** moto:bit carrier board
- **2x** Wheels
- **1x** Assembled Shadow Chassis
- **2x** Hobby Gear Motors
- **1x** 4xAA Battery Holder
- **4x** AA Batteries *(Not Included with Kit)*

Didn't get the kit? Have no fear! Here are the parts you will need to complete this experiment. You may not need everything though depending on what you have. Add it to your cart, read through the guide, and adjust the cart as necessary.

Hobby Gearmotor - 140 RPM (Pair)
◉ ROB-13302

USB micro-B Cable - 6 Foot
◉ CAB-10215

micro:bit Board
◉ DEV-14208

SparkFun moto:bit
◯ DEV-14213

Wheel - 65mm (Rubber Tire, Pair)
◉ ROB-13259

Battery Holder - 4xAA to Barrel Jack Connector
◉ PRT-09835

Shadow Chassis
◉ ROB-13301

Panasonic Alkaline Battery - AA
◉ PRT-15201

Suggested Reading

What is Electricity?

We can see electricity in action on our computers, lighting our houses, as lightning strikes in thunderstorms, but what is it? This is not an easy question, but this tutorial will shed some light on it!

Accelerometer Basics

A quick introduction to accelerometers, how they work, and why they're used.
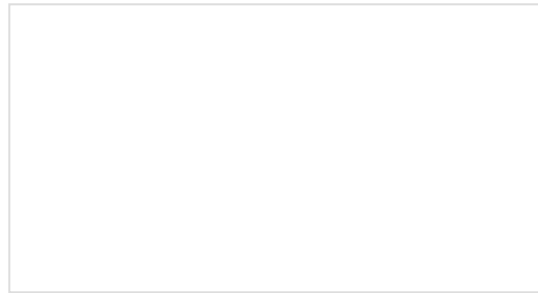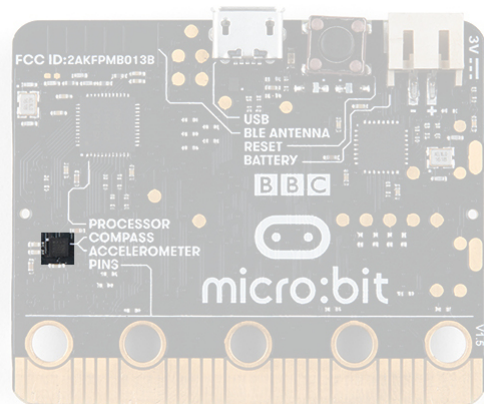
Getting Started with the micro:bit

The BBC micro:bit is a compact, powerful programming tool that requires no software installation. Read on to learn how to use it YOUR way!

Introduction to the Accelerometer

On the back of the micro:bit you can see a number of small chips. One of them is the accelerometer. The micro:bit has an onboard accelerometer that measures gravitational force. Depending on the version that you have, the accelerometer and compass can be on separate ICs or combined into a single IC.
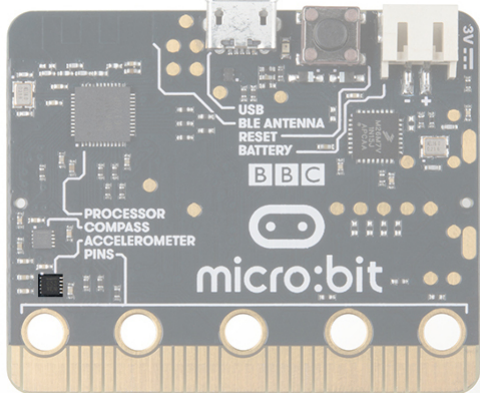
|	
v1.0 w/ Accelerometer on Separate IC	*v1.5 w/ Combined Accelerometer and Magnetometer*

Note: For more information about the motion sensor hardware change, check out this article from micro:bit support.

MICRO:BIT SUPPORT: MOTION SENSOR HARDWARE CHANGE

An accelerometer is a sensor that measures the gravitational forces pulling on it in all three dimensions of the chip's X, Y and Z axes.
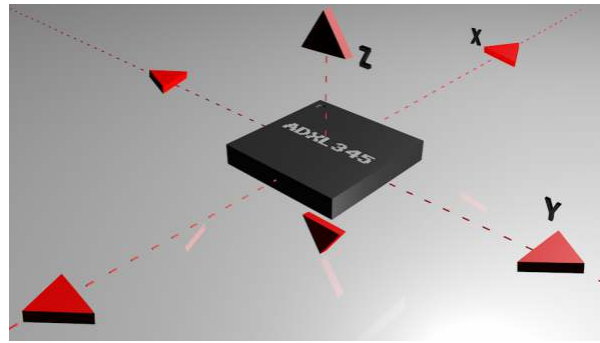
Visualization of a Common Accelerometer (ADXL345) with Three Axes

Accelerometers are devices that measure acceleration, which is the rate of change of the velocity of an object. They measure in meters per second squared (m/s2) or in G-forces (g). A single G-force for us here on planet Earth is equivalent to 9.8 m/s2, but this does vary slightly with elevation (and will be a different value on different planets due to variations in gravitational pull). Accelerometers are useful for sensing vibrations in systems or for orientation applications. Vibration is what we are looking for!

Hardware Hookup

You have had the hardware hooked up the whole time! It is on the micro:bit! Feel free to move on.

Running Your Script

Be sure to add the moto:bit package as instructed in the Installing the moto:bit Package in MakeCode section of this tutorial.

Now, you can either download the following example script below and drag and drop it onto your micro:bit, or use it as an example and build it from scratch in MakeCode.

▶ Simulator   🔧 Blocks   {} JavaScript   ☑ Edit

```javascript
onGesture(Gesture.ThreeG, () => {
tobit.setMotorSpeed(Motor.Left, MotorDirection.Reverse, 100)
tobit.setMotorSpeed(Motor.Right, MotorDirection.Reverse, 100)


onGesture(Gesture.ScreenDown, () => {
tobit.enable(MotorPower.Off)


onGesture(Gesture.ScreenUp, () => {
tobit.enable(MotorPower.On)
```
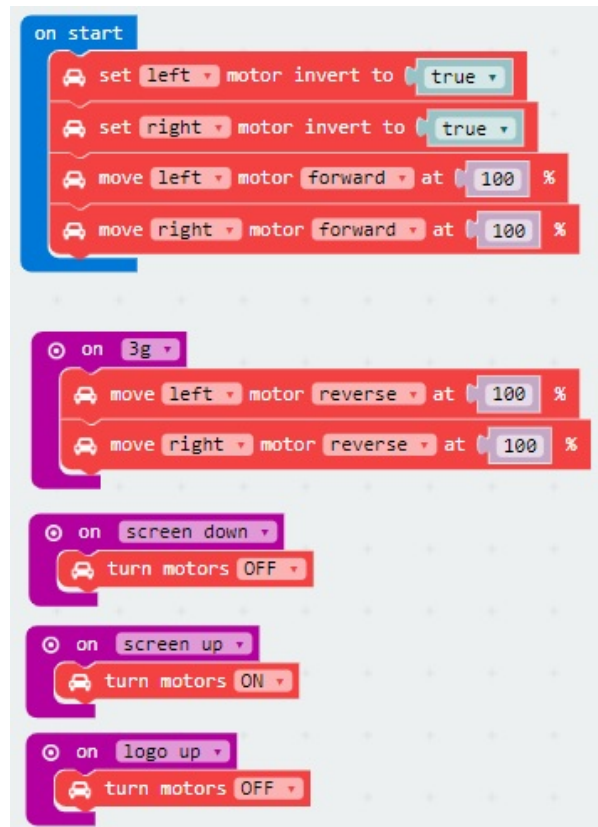
```
onGesture(Gesture.LogoUp, () => {
tobit.enable(MotorPower.Off)
```

Code to Note

Let's take a look at the code and what to expect.

On [Accelerometer]

The micro:bit has a built in accelerometer that measures the gravitational forces that are acting upon it. With other microcontrollers this is sometimes hard to use and make sense of the data you receive from the sensor. But, MakeCode has simplified that for you in an event block under the Input drawer (you will see it as `On Shake`) that allows you to pick a number of different force levels, orientations and even patterns (like shaking it). We use this to detect a number of orientations that you can put your robot in. For example the motors will only be `ON` in the screen up orientation.

If you tap the robot hard enough it will run / drive away from you. If you flip it over the motors will turn off. Same thing holds true if you place it on its tail with the micro:bit pointing up.

Try playing around with the different events that you can trigger with the accelerometer. Can you get it to detect bumping into an object like the wall and have it respond???

What You Should See

Once your code is loaded to your micro:bit and your motor switch is set to "RUN MOTORS." Your robot should not do anything at first. Give it a good tap / smack on top of the micro:bit. If you apply enough force to it your robot should run away. To stop your robot either flip the robot over to its back the motors should stop, or pick it up and

set the robot on end with it sitting on its tail with the micro:bit logo pointing up. To run the program again, return the robot to its wheels right side up and smack it again.

micro:bot kit: Exp4
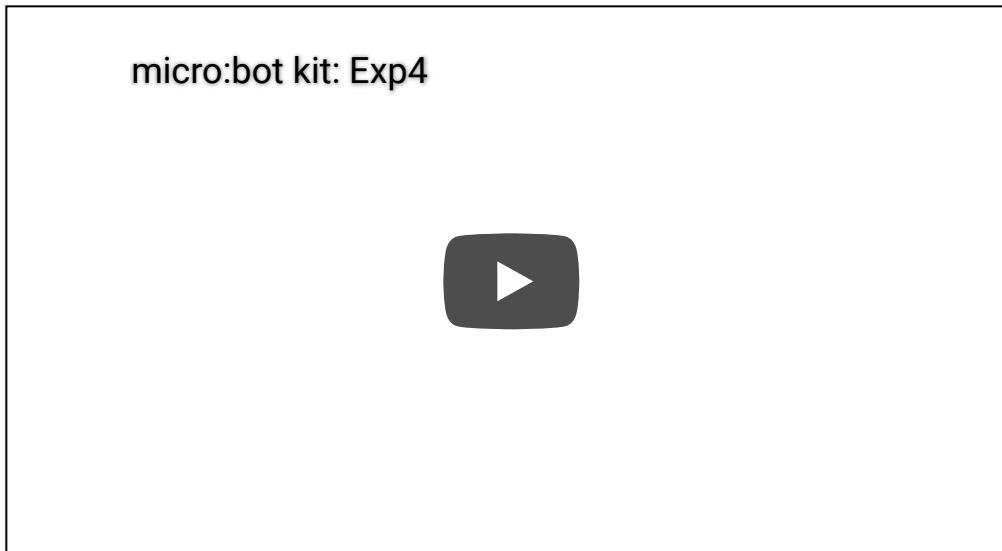
> **Go Further:** You can use the idea of detecting a bump in different ways. Try to write a program that giving your robot a nudge or tap starts it to drive forward for a bit, then you need to nudge it again a bit.

Troubleshooting

- **Robot Doesn't Move** - Double check to make sure you have uploaded the new code to your board. Also, make sure your motor switch is set to the "RUN MOTORS" position.

- **Moving in Reverse?!** - There are two options if you notice the micro:bot moving in the wrong direction. As explained above, you can flip the wiring of your motors or use the `set ____ motor invert to ____` block in your `on start` block to change what is forward vs. reverse.

Experiment 5: Controlling a Servo - Battle Bot

Introduction

Your robot should do something more than just drive around! Maybe control a small robotic arm, a gripper or even a weapon of some kind. In this experiment you will integrate servo motors with your robot to build a Battle Bot with Jousting Lances. This experiment combines all the elements you've learned so far in this tutorial.

Parts Needed

You will need the following parts:

- **1x** micro:bit board *(Not Included with Kit)*
- **1x** Micro-B USB Cable *(Not Included with Kit)*
- **1x** moto:bit Carrier Board
- **2x** Wheels
- **1x** Assembled Shadow Chassis
- **2x** Hobby Gear Motors

- **1x** 4xAA Battery Holder
- **4x** AA Batteries *(Not Included with Kit)*
- **3x** Analog Line Following Sensors
- **3x** 3-Pin Jumper Wires
- **2x** Hobby Servos

Didn't get the kit? Have no fear! Here are the parts you will need to complete this experiment...

Servo - Generic (Sub-Micro Size)
◉ ROB-09065

Hobby Gearmotor - 140 RPM (Pair)
◉ ROB-13302

micro:bit Board
◉ DEV-14208

SparkFun moto:bit
○ DEV-14213

Wheel - 65mm (Rubber Tire, Pair)
◉ ROB-13259

Battery Holder - 4xAA to Barrel Jack Connector
◉ PRT-09835

SparkFun RedBot Sensor - Line Follower
◎ SEN-11769

Shadow Chassis
◎ ROB-13301

Panasonic Alkaline Battery - AA
◎ PRT-15201

Jumper Wire - 0.1", 3-pin, 6"
◎ PRT-10368

You Will Also Need

To complete this experiment, we are going to also need a few more components.

- **2x** Skewers *(Not Included with Kit)*
- **2x** Ping Pong Balls *(Optional, Not Included with Kit)*
- Hot Glue or Tape *(Not Included with Kit)*

Suggested Reading

Hobby Servo Tutorial
Servos are motors that allow you to accurately control the rotation of the output shaft, opening up all kinds of possibilities for robotics and other projects.

Getting Started with the micro:bit
The BBC micro:bit is a compact, powerful programming tool that requires no software installation. Read on to learn how to use it YOUR way!

## Introduction to the Servo Motor

Unlike the action of most motors that continuously rotate, a servo motor can rotate to and hold a specific angle until it is told to rotate to a different angle. You can control the angle of the servo by sending it a Pulse Width Modulation (PWM) pulse train (turning a pin on and off really fast at different intervals); the PWM signal is mapped to a specific angle from 0 to 180 degrees in the servo block with MakeCode.
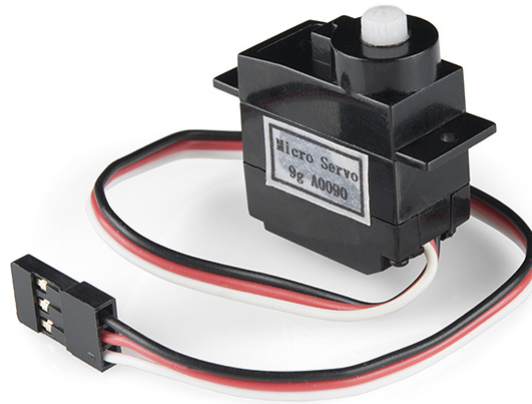
Inside of the servo there is a gearbox connected to a motor that drives the shaft. There is also a potentiometer that gives feedback on the rotational position of the servo, which is then compared to the incoming PWM signal. The servo adjusts accordingly to match the two signals.

Hardware Hookup

In this experiment, you will actually be using two different servo motors to create a Battle Bot! One will be used as a "left" jousting lance and the other as the "right". If you look at the servos, you will notice that they have three wires have different colors; red, black and white. Red is the supply voltage, black is ground and white is the signal. Hook the two servos up to the moto:bit at the two labeled servo ports. Make sure you line up ground to ground.
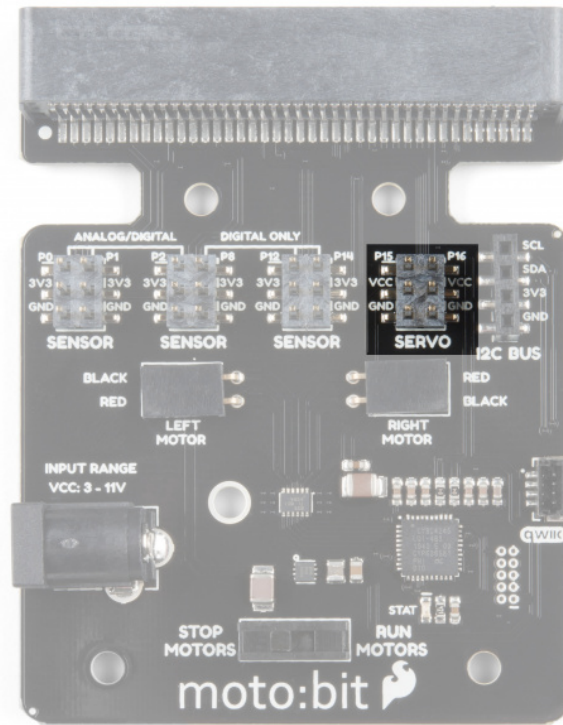
The "left" is connected to pin P15 while the "right" servo is connected to pin P16. Using a bit of hot glue or tape, you can attach the servos to each side of the robot chassis.

Next, you can build the jousting lances by utilizing some grilling skewers! You will only need two that can be cut down to the needed size. You can either glue or tape each skewer to the servo to the supplied attachments. Each servo set comes with a small bag of attachments as seen below. In this experiment, we utilized the double arm micro horn, but any of them will work.

Don't have any skewers laying around? Be creative here!

Bot Assembled with Servos

This Battle Bot utilized electrical tape to mount the jousting lances to the servo motors, and it ended up looking something like this...
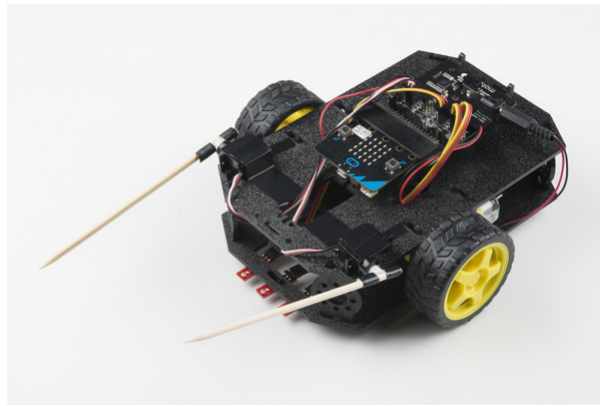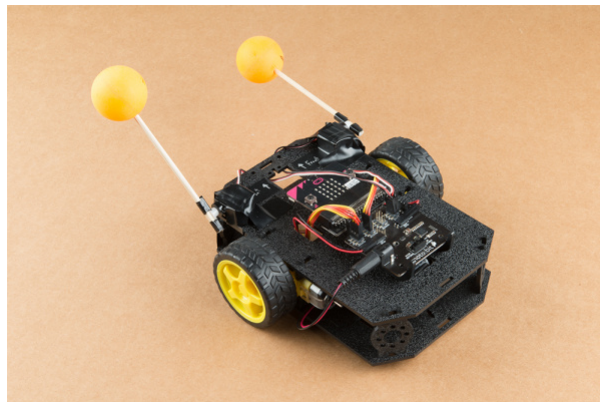
Now, set your contraption aside until you have uploaded your code.

Note: When you think of jousting lances, they can have pointy ends. If you feel the need, grab a set of ping pong balls and poke a hole carefully with the skewers. Secure the ping pong balls to the skewers with some glue. If you need, check with an adult for assistance.

Not as terrifying but safe.

Running Your Script

Be sure to add the moto:bit package as instructed in the Installing the moto:bit Package in MakeCode section of this tutorial.

Now, you can either download the following example script below and drag and drop it onto your micro:bit, or use it as an example and build it from scratch in MakeCode.
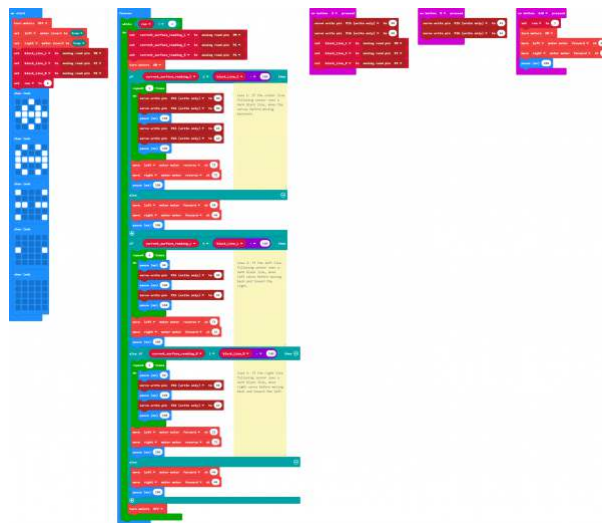
▶ Simulator   🧩 Blocks   {} JavaScript   ↗ Edit

```
on start
    turn motors  OFF ▽
    set  left ▽  motor invert to  true ▽
    set  right ▽  motor invert to  true ▽
```

Microsoft MakeCode | Terms of Use | Privacy | ⬇ Download

Code to Note

Don't be overwhelmed by everything this code has to offer! I know it looks rather menacing, but the truth is that you've already done all of this code in previous experiments.

Having a hard time seeing the code? Open the MakeCode in a separate window to edit or click on the image for a closer look.

Servo Write Pin to

The `Servo Write Pin to` block is found under the Pins drawer and it accepts a pin number that you have a servos signal wire attached to as well as an angle that you want it to rotate to. That's the basics for controlling a servo motor with Microsoft MakeCode!
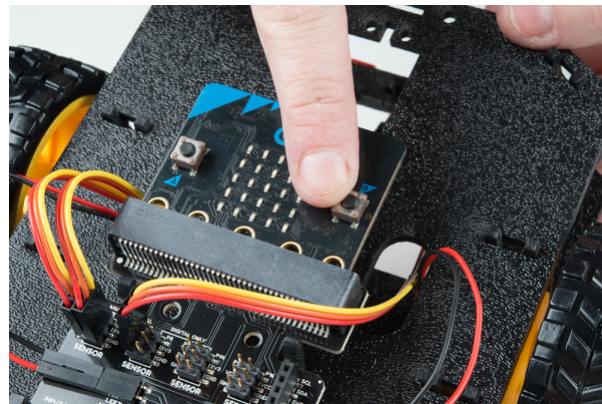
On Button A, On Button B and On Button A+B

In this example we use the `On Button` event blocks a lot. They allow us to break out different functions we want without messy `if` statements.

In this program the `On Button A` event block resets the calibration value for the table surface for the robot to use as a comparison value and also sets the servo positions to 90 degrees.
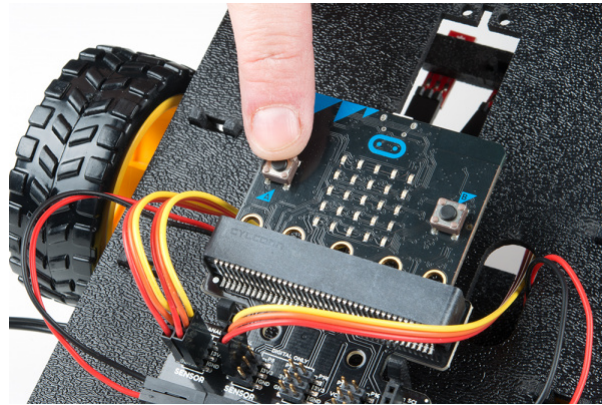
The `On Button B` block is used to set servo position to 45 degrees.

Finally the `On Button A+B` event block is used to run the program.

If / Then Statements

You may notice that there is a few `if / then` blocks inside of the if statement in the `forever` block. This will aid your Battle Bot in making decisions and knowing what it has to do next.

What You Should See

Before running the program, make a thick line (about 12" wide x 6" long) using black electrical tape. This is where your robot will meet with an opposing robot.

When the code initializes, the servo "arms" will move into a starting position and take initial readings for your line sensors to establish a comparison point to the ground environment. Make sure to calibrate it with a black line before starting. Then place each robot at an equal distance away from the line.

Initially, the Battle Bot will not move. Press on buttons A and B at the same time to run the program. The robot will move forward if none of the line sensors detect the black line. If the center line sensor detects a black line, the Battle Bot will immediately start both servos and move backwards. If the left or right line sensors detect a dark surface, the side that detected it will have the corresponding jousting lance start to move while turning away from it. So your Battle Bot is prepared to defend itself from any oncoming situation!

i   b t kit E   5

micro:bot kit: Exp5

[▶]

> **Go Further:** Design two parallel tracks using a black surface. Separate the tracks with a center "wall". Adjust the servos and extend the length of the skewers so that the robots can joust the other robot across the wall. Design and program the robot to follow their respective paths without turning back using the line following sensors. See who can disarm the other robot first.
>
> Or figure out a creative way for it to know when to drop something at a specific location. Could you also program it to pick something up?

Troubleshooting

- **Robot Not Moving** - Make sure your motors are hooked up correctly in the motor ports and your motor power switch is set to "RUN Motors", the battery pack is plugged in, and you have fresh batteries.

- **Moving in Reverse?!** - There are two options if you notice the micro:bot moving in the wrong direction. As explained above, you can flip the wiring of your motors or use the `set ____ motor invert to ____` block in your `on start` block to change what is forward vs. reverse.

- **Servo Moves in the Wrong Direction** - Flip your servos, or change your code so that it rotates the correct direction.

- **Servo Doesn't Move at All!** - Double check your connections of the servos to the moto:bit to make sure that they are hooked up correctly.

Bonus Experiments!

Go further by pairing the micro:bot kit with parts from the micro:arcade kit!
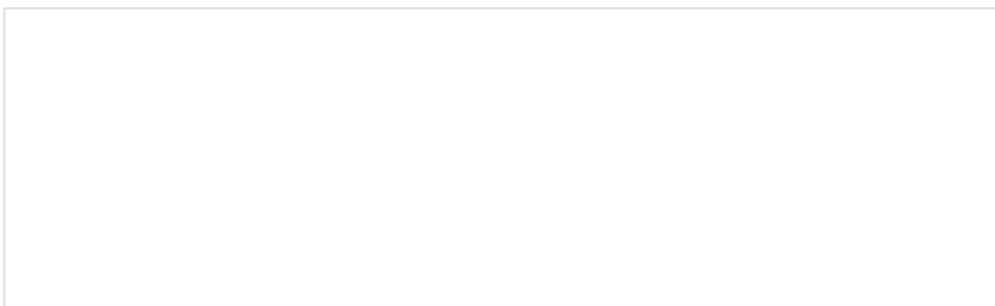
Wireless Remote Control with micro:bit

JANUARY 21, 2019

In this tutorial, we will utilize the MakeCode radio blocks to have the one micro:bit transmit a signal to a receiving micro:bit on the same channel. Eventually, we will control a micro:bot wirelessly using parts from the arcade:kit!

Resources and Going Further

For more information about the moto:bit, check out the resources below:

- Schematic (PDF)
- Eagle Files (ZIP)
- GitHub

We produce a number of other kits and carrier boards that you can hook up to your micro:bit to help take your projects to the next level. Here is some further reading that may help you along in learning more about the world of electronics.

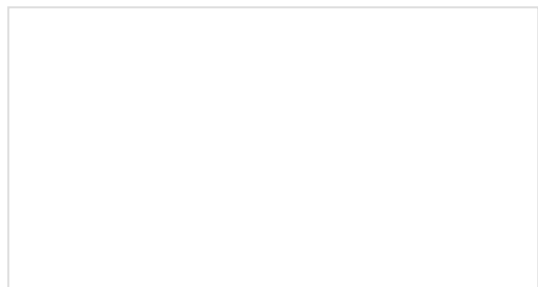For more information on our micro:bit ecosystem, check out these tutorials:
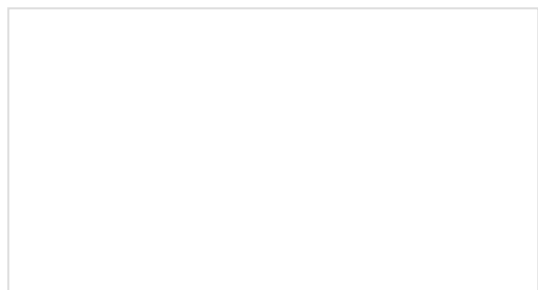
Gator:starter ProtoSnap Hookup Guide
Get started clipping sensors and lights to the micro:bit with the gator:starter gator:board from SparkFun!
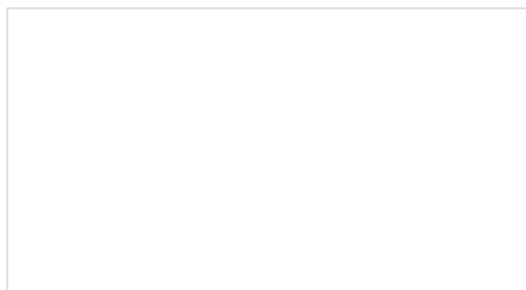
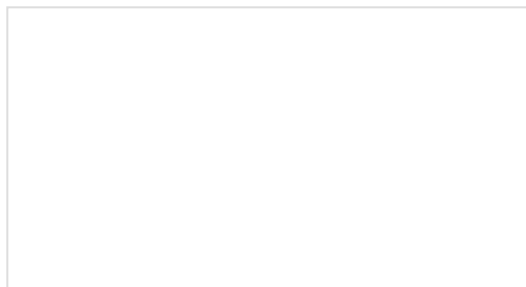### Gator:color ProtoSnap Hookup Guide
Clip some LED's onto your gator:bit with the gator:color.

Gator:control ProtoSnap Hookup Guide
Buttons and switches and inputs oh my! Start adding more control to your gator:bit with the SparkFun
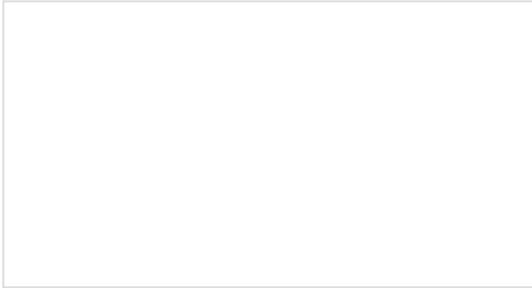
SparkFun gator:bit v2 Hookup Guide
The gator:bit v2 is a breakout board for the BBC micro:bit. The gator:bit exposes almost every pin on the
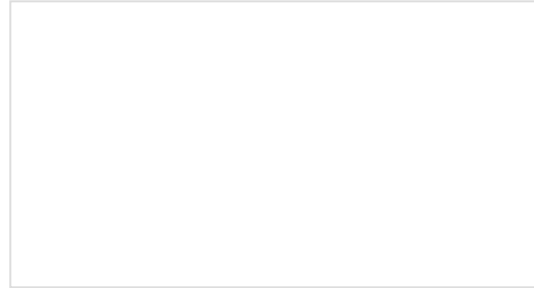
gator:control.

micro:bit to clippable pad with circuit protection. It also has as built-in addressable LEDs and a built-in buzzer.

For more robot fun, check out our these other tutorials outside of the micro:bit ecosystem.

Servo Trigger Hookup Guide
How to use the SparkFun Servo Trigger to control a vast array of Servo Motors, without any programming!

Building an Autonomous Vehicle: The Batmobile
Documenting a six-month project to race autonomous Power Wheels at the SparkFun Autonomous Vehicle Competition (AVC) in 2016.
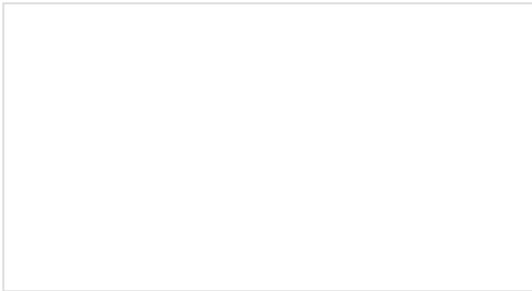
Wireless Remote Control with micro:bit
In this tutorial, we will utilize the MakeCode radio blocks to have the one micro:bit transmit a signal to a receiving micro:bit on the same channel. Eventually, we will control a micro:bot wirelessly using parts from the arcade:kit!
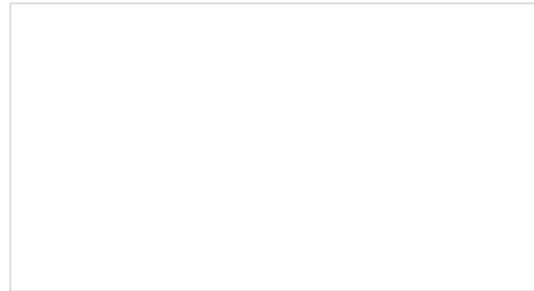
Advanced Autonomous Kit for Sphero RVR Assembly Guide
Get your Advanced Autonomous Kit for the Sphero RVR built up with this hookup guide!